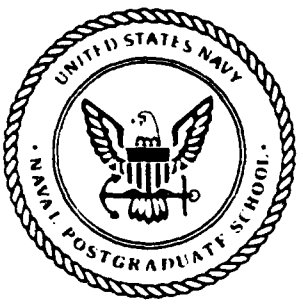
NAVAL POSTGRADUATE SCHOOL
Monterey, California



DISSERTATION

A DATABASE APPROACH
TO COMPUTER INTEGRATED MANUFACTURING

by

Dana E. Madison

June 1988

Thesis Advisor: C.T. Wu

Approved for public release; distribution is unlimited

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A201030

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution in unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A DATABASE APPROACH TO COMPUTER INTEGRATED MANUFACTURING

12. PERSONAL AUTHOR(S)
Madison, Dana E.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
PhD Dissertation	FROM _____ TO _____	1988 June	321

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Integration; Manufacturing; Data Model; Data-oriented

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This work presents a new approach to the integration of manufacturing activities. The manufacturing environment has capitalized on the use of automation to evolve to a highly specialized state characterized by heterogeneous systems providing computer support to the various activities. Conventional approaches to integration assume that these activities must continue to exist in their current relationships. We use a database approach to the integration problem which removes the traditional boundaries between activities. We develop a data model which captures more of the semantics of the manufacturing environment than existing models and allows us to take a data-oriented perspective of the activities it encompasses. We also show how the use of the data-oriented approach provides for integration of these activites and reduces the complexity of the manufacturing environment.

Keywords: high level interface, (KR) (—

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. C. T. Wu	22b. TELEPHONE (Include Area Code) (408) 646-3391 22c. OFFICE SYMBOL Code 52Wq

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited

A Database Approach to Computer Integrated Manufacturing

by

Dana E. Madison
Major, United States Army
B.S., State University College at Brockport, NY, 1972
M.A., State University College at Brockport, NY, 1975

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

Author: Dana E Madison
Dana E. Madison

Approved by:

D. K. Hsiao
Professor of Computer Science

R. B. McGhee
Professor of Computer Science

J. R. Ward
Professor of
Electrical and Computer Engineering

B. O. Shubert
Associate Professor of
Operations Research

C. T. Wu
Associate Professor of Computer Science
Dissertation Supervisor

Approved by:
V. Y. Lum, Chairman, Computer Science Department

Approved by:
Kneale T. Marshall, Acting Academic Dean

ABSTRACT

This work presents a new approach to the integration of manufacturing activities. The manufacturing environment has capitalized on the use of automation to evolve to a highly specialized state characterized by heterogeneous systems providing computer support to the various activities. Conventional approaches to integration assume that these activities must continue to exist in their current relationships. We use a database approach to the integration problem which removes the traditional boundaries between activities. We develop a data model which captures more of the semantics of the manufacturing environment than existing models and allows us to take a data-oriented perspective of the activities it encompasses. We also show how the use of the data-oriented approach provides for integration of these activities and reduces the complexity of the manufacturing environment.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to express my thanks to my advisor, C. Thomas Wu, whose support and guidance was invaluable to me. The suggestions received from Professor David Smith of the Mechanical Engineering Department helped me gain insight from an engineer's perspective and are highly appreciated. The guidance I received from the members of my Doctoral Committee, Professor David Hsiao, Professor Robert McGhee, Professor Bruno Shubert, and Professor John Ward, strengthened the presentation of my research work in this thesis.

I would also like to express my appreciation to my family for their love and unfailing support. In particular, I am grateful to my wife Linda for her encouragement throughout my course of study. Dr. and Mrs. Paul deR. Kolisch deserve special recognition for their love and inspiration over the past 22 years.

I would also like to thank the U.S. Army Medical Department for their financial support of my research and studies at the Naval Postgraduate School.

I. INTRODUCTION

A. BACKGROUND

Computers have established themselves as powerful tools in the attempt to drive down costs and improve efficiency in the manufacturing environment. To date, their application has been piecemeal, they are not part of a concerted effort to integrate business activities [Ref. 1]. The goal of this research is to examine the potential for use of data modeling aspects of database technology in integrated manufacturing, in particular the data interactions which form the basis of an integrated system. Alternative approaches to integration which utilize advanced database and artificial intelligence technologies are explored. The focus of the research is on developing a data model for the approach we deem most feasible. The research includes an extensive study of the current manufacturing functions and semantic data modeling.

B. THE PROBLEM

The introduction of intelligent integrated automation into the factory has the potential to increase efficiency and optimize utilization of resources, the two most important concerns of a manufacturing company trying to keep up with the pace of a rapidly changing marketplace. Within the past decade, product development has evolved from a simple communication process between design engineer and mechanic to a complex system utilizing highly specialized personnel, state-of-the-art automation and communications technology and highly sophisticated manufacturing tools. As businesses grew, tasks were divided up and allocated to people with special skills. The resulting improvement in efficiency was offset by the creation of more complex systems for moving materials and information, causing greater administrative overhead. More work was created in the control and management of the resulting complexity. Few people had insight into more than just a small part of the manufacturing process. Manufacturing

problems were treated singularly rather than being viewed in the context of the entire process, which resulted in the introduction of automation to improve the performance of a small, often self-contained area within a company. The overall effect on performance generally fell short of the potential that could have been achieved by looking at the process as a whole from the start. The fact that automation was introduced provided an opportunity to simplify the overall functions of the manufacturing process. Instead, it was treated as a simple machine replacement process. Another problem inherent in those companies was the *Just in Case* philosophy of production which further complicated the overall system because of the need for buffering inventory at every point where production could be interrupted. Raw materials, material in progress, work in progress, finished parts, and finished products were held just in case interruptions occurred. Managers recognized that tighter control over manufacturing operations was necessary to survive in a competitive environment characterized by rising salaries, falling prices, and diminishing market shares. This realization led to the use of computers to support various functions across the manufacturing spectrum in hopes of achieving control. [Ref. 1]

The use of computers made the concept of dynamically programmable manufacturing tools a reality. Paper-tape driven numerical control (NC) machines logically evolved into computer numerical control (CNC) machines. Design functions became highly specialized and efficient when augmented with Computer Aided Design (CAD) technology. Manual accounting and inventory systems were replaced by management information systems to increase the management span of control over the rapidly expanding business. The result of this massive application of computers was substantial improvements in productivity, quality, cost reductions, and other factors vital to achievement of business goals, and an automation program which was highly fragmented and difficult to control [Ref. 2].

Many of the automated functions were supported by mutually incompatible computers, control devices, and automated machines, most of which could not communicate or exchange data with other systems. Thus, the order flow from customer to shipping dock and the information flow between engineering and manufacturing were

severely fragmented. The solution to restoring the communication and material flows involves the integration of these currently heterogeneous systems using a concept called *Computer Integrated Manufacturing (CIM)* [Refs. 1, 2]. Using this concept, which will be discussed in detail in Chapter III, links are formed between existing islands of automation, gradually evolving towards a totally integrated system.

The alternatives to integration presented here all focus on implementing the CIM concept. Once a manufacturing company makes the decision to implement Computer Integrated Manufacturing technology, the implementation strategy must be determined. The complexity of the manufacturing process dictates that the integration implementation be modular and phased in from the lowest level of operation. In some cases, low-level functions will need to be completely redefined in order to take advantage of CIM technology. The company will experience the initial turmoil caused by changes in operating procedures, the learning curve for new skills, and the anticipation of impending control.

On the other hand, the company will have better control of information and will make better decisions as a result. The availability of relevant information will permit employees' time to be spent more productively and use of other manufacturing resources will be optimized as well.

The successful integration of product design and manufacturing functions requires a complete understanding of the relationships of data produced and used throughout the product life cycle and some mechanism to translate product design data into a form which is useful in the manufacturing process.

A major difference in our work from that of previously published CIM data modeling work is that the modeling technique we propose is capable of describing the structural aspects of products as well as their production processes. This uniformity will enable us to integrate different manufacturing functions in a clear and natural manner. Our approach can be characterized as *data-oriented* since we identify the data requirements of various manufacturing functions and attempt to create a common data manager for them, whereas other efforts can be characterized as process-oriented, since

they attempt to integrate different functions by creating an interface between them, assuming that the data management is effectively handled by the relational DBMS. But the relational DBMS is not a panacea. The traditional relational data manager is simply not capable of expressing complex objects and relationships that exist in the manufacturing environment.

We divided the basic manufacturing activities into four simple stages based on the type of function they performed (see Figure 1). We then proceeded to identify the data requirements of each stage. The benefit of our approach is clear: theirs is evolutionary while ours is revolutionary. In our opinion, there is nothing to be gained by evolving the currently highly fragmented state of manufacturing. It is much more beneficial to attack the problem from a fresh viewpoint.

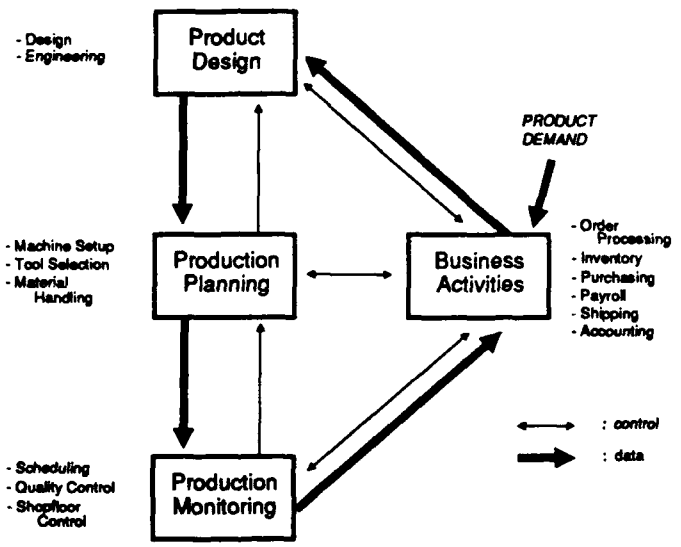


Figure 1. Stages of Manufacturing

This research in data modeling for CIM is just the beginning of an even larger effort to develop an advanced database management system using database engineering techniques [Ref. 3]. This database management system will be able to handle advanced application areas such as tactical weapons systems, industrial manufacturing systems, and

integrated corporate information systems. In current practice, the data for these applications are either handled manually or by a specialized file manager. The requirements for an advanced database management system have been identified in [Ref. 3] and provide direction for the overall research effort.

In this dissertation, we will categorize the different approaches to integrating manufacturing functions and discuss two of these approaches in detail. We will describe the data model we have developed to achieve integration, and describe how this data model supports the manufacturing environment.

II. STATE-OF-THE-ART MANUFACTURING

A. INTRODUCTION

The role of automation in the operations of a manufacturing company can be best portrayed by describing the various functions and activities involved in the design and development of a product. The functions and activities comprise the *product life cycle.* Figure 2 depicts the product life cycle, as formulated in this thesis. We view the life cycle as a series of activities, each interacting with one or more other activities in the cycle.

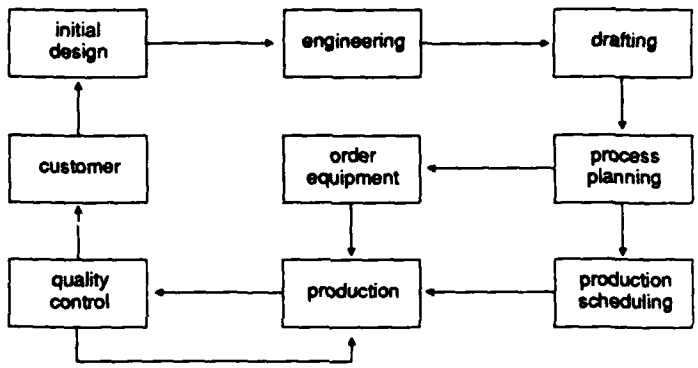


Figure 2. Product Life Cycle

Input to the cycle consists of information about prospective markets and customer desires, also known as the *demand* for the product. It is this demand which drives the decision-making process to determine in what ways the product life cycle will be activated and controlled.

The activities involved in the product life cycle can be further broken down into the basic processes performed within a factory. These include design engineering, process planning, NC machine programming, robot programming, quality control, shop floor management, marketing, sales estimating, order processing, master scheduling, material

requirements planning, plant maintenance, shipping, inventory management, purchasing, and accounting. These processes have been grouped in many different ways forming the functions known as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Aided Process Planning (CAPP), Group Technology (GT), Flexible Manufacturing Systems (FMS), and others. In forming these groupings, the basic processes have not been treated consistently. In some cases material requirements planning is considered part of CAM, others consider it part of CAPP, still others consider it the hub around which the other processes revolve. One of the major problems associated with integrating manufacturing functions (or processes) is that the definition of what constitutes a function is not standard within the manufacturing industry. We will provide definitions for the functions which are most prevalent in our research.

Figure 3 shows the basic processes partitioned into CAD, CAM, FMS, and Business. Our discussion of the manufacturing environment will assume the groupings shown in the figure.

B. COMPUTER AIDED DESIGN

1. The Design Process

The design process starts with the definition of a need which can be satisfied by some product. The definition of this need may involve many people and a lot of time, or may be developed by one person in a short period of time. A general concept of a product is formulated and refined from this definition, eventually producing a specification for the product. Figure 4 illustrates this process.

Note that at any point in the design process, the next step could be to go back to a previous step, reformulate, and work forward again. Design is much more than a simple serial process since each step depends on the result of the previous one and may in fact change the previous one. In trying to formulate a general concept of a solution, it is frequently the case that the need is not well-defined. The possible infeasibility of a design has to be considered in the formulation of the general concept and in the development of

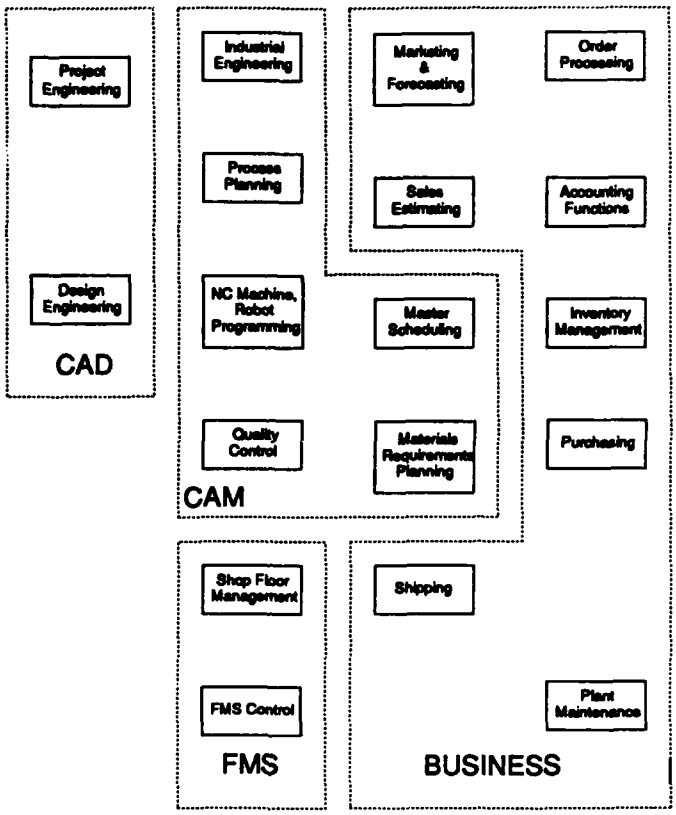


Figure 3. Basic Manufacturing Processes

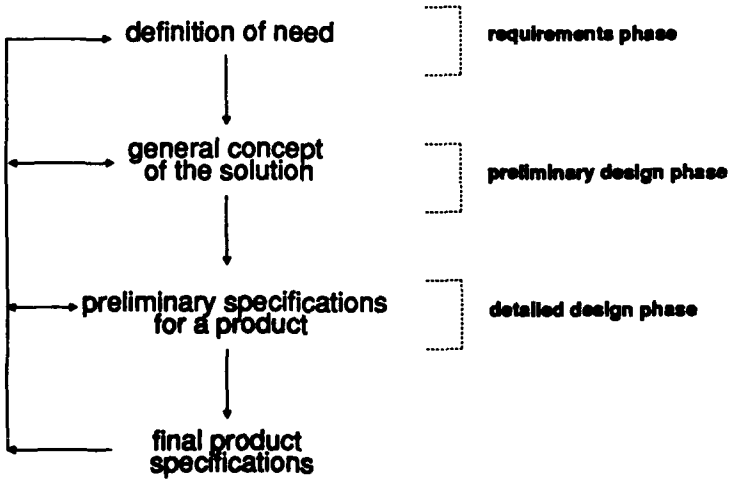


Figure 4. The Design Process

specifications. These, and other unforeseen events, cause product design to be an iterative process.

The design process can also be described in terms of phases, also shown in Figure 4. The initial definition and subsequent modification of the need is done in the requirements phase. The preliminary design phase produces a set of potential solutions to the requirement and determines the best alternative which will establish the direction for the detailed design. The detailed design uses the general concept from the preliminary design phase to synthesize the assemblies and components eventually leading to detailed specifications for a product. These specifications include choices of materials for the individual parts, tolerances, and detailed engineering drawings of the product. Again, the phases are strongly interconnected and interdependent upon each other.

> The design process describes the gathering, handling, and creative organizing of information relevant to the problem situation; it prescribes the derivation of decisions which are optimized, communicated, and tested or otherwise evaluated; it has an iterative character, for often, in the doing, new information becomes available or new insights are gained which require the repetition of earlier operations. Some of the operations are qualitatively logical in character, like reasoning from verbal propositions; some are based on subjective evaluations, as in comparing or combining unlike values; many are amenable to quantitative analysis and to computer applications, as in optimizing an analytically formulated representation of a problem solution. For the most part the techniques associated with each operation in the design process are of such great generality that their usefulness is not limited to any particular step. [Ref. 4]

2. Use of Automation in Design

The availability of Computer Aided Design (CAD) has enhanced the design process by providing design engineers with tools for generating new designs from scratch and modifying existing designs. The existing designs can be located and modified to accommodate new requirements or refined definitions of existing requirements. The result can then be analyzed mathematically to check load factors, stress, etc. A major advantage in the use of CAD technology for the design process is the increased productivity of design engineers due to shortened development time for a product. In addition, the automation of design data has tremendously increased its reliability and reusability.

3. Classification of CAD Systems

One way to classify CAD systems is by the way they represent design objects. Three alternative geometric representations are the two-dimensional drafting systems, and the wire-frame, surface, and solid three-dimensional models. These representations are stored as *computer internal models* [Ref. 5], whose complexity varies from one representation to another. These computer internal models can be structured as a matrix where the types of generated model data (e.g., cylinder, cone, prism) are listed in rows and the computer internal representation (data structure), in columns. A significant characteristic of a computer internal model is its parametric capability to change the dimensions for a specified object while leaving the object's topology (general shape) intact.

Two-dimensional drafting systems are basically automated drafting board systems which display a two-dimensional representation of the object being designed. Engineers using this type of system generally develop a line drawing and produce a high quality output using a pen plotter. While these systems do improve the productivity of the designers who use them, they only produce two-dimensional drawings of three-dimensional objects and it is up to the engineer to read the drawings and infer the three-dimensional shape from them. An additional problem often arises because the two-dimensional drawings are ambiguous. Most two-dimensional drafting systems use a form of modeling called *wire-frame modeling*, so called because the edges of an object are shown as lines and the image of the object appears to be a frame made out of wire. A predominant limitation of a wire-frame model is that all the lines that define the edges and contoured surfaces of an object are shown in the image so that the lines representing the edges at the rear of the object show right through the foreground surfaces. This limitation introduces even more ambiguity into the image interpretation problem. Systems which feature *hidden-line removal* seek to eliminate this ambiguity by removing the hidden background lines in the image.

Three-dimensional wire-frame modeling systems overcome some of the limitations of the two-dimensional systems by allowing an engineer to create a full

three-dimensional model of an object rather than a two-dimensional illustration. The system can automatically generate orthogonal views (much like what the two-dimensional system would have produced), perspective drawings, and even close-ups of detailed portions of an image. While three-dimensional systems have these capabilities, they generally still have the hidden-line removal problem.

Surface models represent the vertices and edges of an object in much the same way as wire-frame models, but also include *polygonal faces* of three-dimensional objects and allow the properties of those faces to be specified. These models are more complex than wire-frame models to implement, but produce much more realistic images with the use of hidden-line removal, colors and textures for the surfaces, shading, and cast shadows.

The most complete kind of three-dimensional model is the solid model, with images composed of objects which appear solid to the viewer [Ref. 6]. Like the surface models, the solid models use color, texture, and shading to make the images appear more realistic, which decreases the likelihood of misinterpretation. The potential for solid modeling applications is driving the graphics technology to produce better and faster hardware and software to provide these capabilities [Ref. 7]. One basic approach to solid modeling is the *constructive solid geometry* (CSG) approach, also called the building block approach [Ref. 8]. In this approach, the engineer builds a model out of solid graphic primitives such as rectangular blocks, cubes, spheres, cylinders, and pyramids. An advantage of this approach is the ease in which a precise solid model can be constructed out of the primitives by adding, subtracting, and intersecting the components.

4. Interfacing CAD Systems

A lot of effort has been put into the development of standard interface specifications to improve communication of product design data between and within systems. Included in these specifications are the *exchange format specifications*, the *geometric modeling interface specifications*, and the *database interface specifications*. Two projects involving exchange format specifications are the *Initial Graphics Exchange Specification* (IGES) [Ref. 9] and the *Product Definition Data Interface* (PDDI) [Ref. 10].

The IGES specification is considered an industry standard for the exchange of data between systems produced by different vendors. The PDDI specification, sponsored by the US Air Force, provides information such as tolerances, features, geometry, topology, and part control needed by manufacturing functions.

The major geometric modeling interface specification project is the CAM-International sponsored *Application Interface Specification* (AIS) [Ref. 11]. Their objective is to provide an interface between functional application programs and constructive solid geometry modeling systems.

One of the database interface specification projects was done at Rensselaer Polytechnic Institute, Troy, NY, as a Ph.D. thesis involving the development of a language for the description of data exchange between heterogeneous CAD databases [Ref. 12].

C. COMPUTER AIDED MANUFACTURING

Consider Computer Aided Manufacturing (CAM), consisting of industrial engineering, process planning, numerical control (NC) machine and robot programming, quality control, master scheduling, and material requirements planning, as shown previously in Figure 3. We will concentrate our discussion on the functions most likely to be affected by the introduction of automation, namely process planning, NC programming, master scheduling, and material requirements planning. Information on the other aspects of CAM can be found in [Ref. 13].

1. Process Planning

The production cycle begins with the planning of production processes and determination of production conditions for machine operations. Traditional process planning is an industrial engineering activity which is performed after product design and before production. Engineers examine a bill-of-materials (BOM) and the design specifications and determine which operations are to be performed for each part on the BOM, which machine will be used for each operation, and the details associated with

tooling and other production processes. The results of the process planning function are used by production personnel to manufacture the product. Since process planning is the basis from which other manufacturing decisions are made, any error made at this point will be propagated and compounded in the subsequent manufacturing functions. Process planning is typically performed by the best manufacturing engineers, those with the most experience. The seniority of these engineers poses a problem because retirements frequently exceed recruitments for process planning positions, creating an ever-increasing gap between the number of process planning jobs and the number of qualified engineers available to fill those jobs.

The automation of process planning functions has been hampered by a limited understanding of the skills used by human planners and the fact that existing tools to support process planning are only partially successful. Further problems occur because of the dynamic nature of process planning functions; inputs change frequently, outputs serve many types of users. Most automation attempts focus on process planning as an interface between product design and production, its traditional role [Refs. 14, 15]. These systems mimic the way process planning was previously done, replacing human process planners by automated and sometimes expert systems. The integration of design and manufacturing functions in this way further compounds the *islands of automation* problem associated with the introduction of automation into the factory [Ref. 16]. Other attempts to use artificial intelligence have to deal with a solution space which is well-populated because of the number of parameters involved in the manufacturing of a product. In addition, the solution space is discontinuous; a change in a tolerance may or may not cause a change in the process plan and a change in material type could have the same effect.

One of our objectives in this research is to provide a mechanism which will ameliorate some of these problems. We will show that the data model which we propose for the product design environment can be adapted nicely to the process planning environment, which is the heart of computer aided manufacturing.

Automation has been used to try to create an efficient PP system. The solutions obtained fall into two groups, based on the way they derive a process plan. The first, the *variant system*, uses parts classification with group technology to create a new process plan from an existing template [Ref. 17]. The template represents a standard process plan for a group of products which have been classified into the same family. When the production requirements for a product differ from the general requirements of the family to which it belongs, the template is modified to create the process plan for that product.

An alternative approach is to generate an individual PP for each product from scratch, known as the *generative* approach [Refs. 17, 18]. In a generative system, the knowledge about how products should be manufactured is stored and used with algorithms to create a process plan. A generative system normally starts with design information about the components of a product, information about material types and their usage in manufacturing, and synthesizes an optimal PP. In general, systems using the generative approach are limited to a small range of manufacturing processes because of the complexity of the knowledge involved.

The major disadvantages of the variant method as compared with the generative method include difficulty in accommodating the numerous combinations of geometry, size, precision, material, quality, and shop loading, and the enormous on-line database requirements to accommodate the stored plans.

2. NC Programming

The concept of controlling machines by programming them with a series of alphanumeric codes emerged in the U.S. in the early 1950's [Ref. 13]. The concept is fairly simple: control the machines using numbers to represent a desired function (e.g., switch on the spindle, retrieve tool X, rotate the robot wrist by 30 degrees, etc.) on a predefined coordinate system. These computer-controlled machines can be classified into categories based on the method by which they process workpieces.

Point-to-point controlled machines move a slide to a discrete coordinate point and this movement occurs with the machine tool disengaged, i.e., the tool is never in contact with the workpiece while the slide is in motion. Typical examples of this type of machine include an NC coordinate drilling machine with an NC controlled xy table.

In contrast to point-to-point machines, *straight-line control* systems allow the tool to be in contact with the workpiece while the slide is moving, but the movements are always parallel to the axes of the machine. A typical example of this type of machine is an NC milling machine with a traverse table.

Continuous path controlled machines follow a mathematically prescribed path where anywhere from two to six axes are controlled simultaneously while the tool is in contact with the workpiece. This type of machine normally employs sensors to monitor and control the operation while enhancing the safety and reliability of the machine [Ref. 19]. A typical example of this type of machine is an NC spray painting robot.

3. Master Scheduling

Scheduling is a process that relates specific events to specific times and/or time periods. This relation involves the sequence and timing of assigning resources (i.e., machines, employee, etc.) to specific orders for products. Scheduling gets its prominence from the effect that misutilization of resources and missed due dates have on the profitability of a manufacturing company. Due to increasing costs and shrinking market shares, a lot of emphasis is being put on this aspect of manufacturing.

The scheduling of the processes required for the manufacturing of products involves simultaneous consideration of the processes to be performed and the resources they require. The determination of the appropriate processes for a given product is made during the process planning phase of product development. The scheduling problem is to utilize resources as efficiently as possible while completing all product development as closely as possible to their due date, minimizing in-process inventory. Resources to be scheduled include machines, tools, raw materials, materials in progress, storage facilities, transportation facilities, and labor. The labor resources will generally have variable

capacities while the machine resources will have relatively fixed capacities. The scheduling problem assumes each resource has an operating capacity at each point in time.

Several factors influence the approaches to solving the scheduling problem. First, *combinatorial complexity* occurs in scheduling because of the large number of schedules which assign a set of jobs to a set of resources. The scheduling problem is a classical application area for operations research optimization techniques and is generally known to be NP-complete [Ref. 20]. That is, even if an optimal solution could be found, the amount of time required to compute the solution would make this type of scheduling impractical. Hence, the operations research approaches generally produce sub-optimal results.

Secondly, *uncertainty* is prevalent in the scheduling problem because of the unforeseen events, such as machine failures, which would disrupt a schedule and create a whole new scheduling problem. Even though all of the disruptive events can't be predicted in advance, some approaches build slack time into the schedule to allow for some of them.

The current approaches to the scheduling problem can be classified into several categories, including *opportunistic*, *optimization-based*, and *expert system-based*. We will briefly describe these approaches to provide a means for comparison with our approach.

In the opportunistic approach, the number of possible schedules is reduced prior to execution using knowledge of the current operating conditions. Once schedule execution begins, choices are made among the alternative partial schedules so as to maintain steady progress while maintaining the greatest number of future choices, thus preserving flexibility in the system. In [Ref. 21], off-line reasoning is used to select an appropriate group of partial schedule orders to be passed on-line when schedule execution begins. Combinatorial complexity is managed by not selecting or pruning a particular schedule from the set of possible schedules until there is good reason to do so. Uncertainty is managed by preserving flexibility with the idea that if enough options can

be maintained, there will always be a way to make progress in spite of the unforeseeable problems.

The optimization-based approach formulates the scheduling problem as a linear optimization problem and solves the problem statically using combinatorial optimization techniques such as dynamic programming and branch-and-bound. The basic problem with this approach is that the optimization techniques are used statically, that is, the entire schedule is optimized for all products concerned using the current operating conditions. When something happens on the shop floor such as a machine failure, a whole new optimization problem exists and the schedule has to be redone.

Expert system approaches generally use heuristics which reduce the number of possible schedules and make the scheduling problem solvable. Alternative strategies include the use of scripts which contain the appropriate operators for a particular scheduling situation, and constraint-driven techniques, where domain knowledge is represented as constraints which bound and guide the search for a feasible solution. This approach continually reduces the number of possible schedules until one acceptable schedule remains. If an acceptable solution is not found, constraints are relaxed until acceptable alternatives are produced.

Almost all of the previously mentioned approaches use rules to establish the relative priority of jobs to be processed through a given work center. Some of the rules used include giving priority to the job with the earliest due date, shortest processing time for the work center in question, first-come-first-served, least slack time remaining (time until due date minus process time remaining), and least critical ratio (time until due date divided by process time remaining). No single rule is suitable for all situations, each rule has its merits and drawbacks. For example, using the shortest processing time rule will generally result in the lowest manufacturing lead times and the lowest in-process inventory, but long processing jobs will always lose out, and may never get to the front of the queue without some type of intervention.

4. Material Requirements Planning

The objectives of Material Requirements Planning (MRP) in CAM are to plan and release production orders, focus on orders requiring attention, and ensure that all parts, both manufactured and purchased, are available when the production schedule requires them. MRP incorporates long-range business strategies, short-term tactical plans, master schedules, and feedback on performance [Ref. 22]. The input to the MRP process is the master schedule containing information on the quantities of each ordered product and the dates they are scheduled for production. MRP uses bill of material, inventory status, and order lead time information to generate a more detailed production schedule. As production proceeds, data on the finished products are fed back into the system and the process starts all over again.

D. FLEXIBLE MANUFACTURING SYSTEMS

A flexible manufacturing system (FMS) is a computer-controlled configuration of semi-independent workstations and material handling systems which is designed to efficiently manufacture multiple products at low to medium volumes. A typical manufacturing cell consists of a numerical control (NC) machine, tool machine, a tool magazine, a robot controlled (RC) handling device to refill the magazine, and a system for part supply. The material handling system is responsible for part transportation, raw material and final product transportation, and storage of workpieces, empty pallets, auxiliary materials, waste material, fixtures, and tools. One aim of using FMS technology is to combine the benefits of a highly productive, but inflexible transfer line with a highly flexible, but inefficient job shop.

FMS technology changes the production philosophy of a shop from the traditional *Just-in-Case* to the innovative *Just-in-Time* (JIT) philosophy. The Just-in-Case approach complicates the entire production process because of the need for buffering inventory at every point where production could be interrupted. Raw materials, (raw) material in progress, work in progress, finished parts, and finished products are held "just in case" interruptions occur. The recognition by managers that elimination of the buffered

inventory would significantly cut costs led to the adoption of the Japanese Just-in-Time production system. Other key technologies which similarly reduce costs, enhance productivity, and operate within the FMS framework are *Group Technology* (GT) and *Cellular Manufacturing* (CM).

1. Just-in-Time

The main objective of JIT is the elimination of waste, where waste is defined as any activity which does not add value to the product [Refs. 23, 24]. The types of activities which fall into this category include transporting materials and parts, storing inventory, inspection and quality control (as a separate activity), and machine setup. JIT changes the emphasis in production from producing quantity to producing quality.

A partial solution to the main objective is to minimize manufacturing throughput time. With traditional approaches, production lots spend most of their time in queues, waiting to be worked on [Ref. 25]. Many manufacturers are looking at cellular or group technology concepts to improve material flow and reduce setup time, which will improve throughput.

2. Group Technology

Group technology (GT) is a coding and classification system used for combining similar, often-used parts into families. The use of GT helps to standardize the fabrication of similar parts, allowing them to be retrieved and processed in an efficient, economical way. Depending on the type of GT implementation, parts can be grouped into families in different ways. One common type of GT places parts in a family when they share similarities in their design. Some of the attributes used in this grouping include the part's basic external shape, basic internal shape, length-to-diameter ratio, and material type. A second type of GT uses manufacturing attributes to classify parts. Among the attributes considered in this case are the major process to be performed, minor processes to be performed, machines and tools used, and operation sequence. In this second GT coding scheme, parts whose major process is drilling holes are separated from those whose major process is boring holes.

The use of GT promotes standardization in manufacturing, which eventually translates into improved efficiency and reduced production costs. Jigs and fixtures can be designed to accommodate parts families thereby reducing setup time and costs. Another benefit of the use of GT is the reduction in complexity and size of the parts scheduling problem brought about by grouping parts into families. A 70% reduction in production time, 62% reduction in work-in-process inventories, and 82% reduction in overdue orders has been reported in [Ref. 26]. The time and cost associated with the process planning function itself can be reduced through the standardization achieved as a result of using GT.

3. Cellular Manufacturing

The concept of group technology is closely related to that of cellular manufacturing (CM). In CM, the manufacturing resources are divided into production cells. Each cell is designed to produce a set of parts that require similar machinery, tooling, machine operations, and/or jigs and fixtures. The objective of CM is to go from raw material to finished part within a single cell.

While the concept of CM is theoretically appealing, in practice there may be conditions under which it may be impossible to employ. For example, there are always products to be produced which can't be associated with a specific production cell. In addition, there may be machinery which can't be placed in any one cell due to its general use, such as a spray painting booth. CM technology thus far has been applied to a limited number of applications, the majority of which are chip producing, metal fabricating, and assembly operations [Ref. 27].

E. BUSINESS DATA PROCESSING

The business data processing functions necessary to support the product design and manufacturing processes include customer order processing, production of a bill of materials for each product, capacity planning and control for the shop floor, inventory

control, purchasing, and product costing tasks. Other functions are included as well, but are either well known or well-explained in other references [Ref. 13].

Customer order data is used for long-range planning as well as for material requirements planning. The long-range planning includes providing forecasts for future orders. The short-term planning provides information for capacity planning and scheduling of resources.

Capacity planning provides data on the required machines, personnel, equipment, and parts inventory required to manufacture products. With the JIT philosophy, a primary concern in capacity planning is the amount of inventory to be held. Frequent checks must be made, comparing the actual on hand levels to the planned levels, to maintain control. Again, minimizing inventory levels keeps inventory storage costs, related capital investments, and taxes as low as possible.

A bill of materials is a description of a designed product in which the relationships between components, assemblies, and sub-assemblies are given in the form of a list. The list contains information such as a part number, part description, and quantities of the part for the entire product.

Inventory control is closely related to capacity planning. Inventory items consist of raw materials, work in progress, parts in progress, finished products, and parts purchased from other vendors. Besides keeping inventory levels as low as possible, inventory control includes the administrative aspects of inventory such as recording current stock levels, producing purchase requests, processing customer order and shipment information, forecasting future inventory requirements, and handling the inventory portion of the financial activities (accounts payable, accounts receivable).

Product costing involves determining the cost of every activity related to a given product. Every part of the factory that performs these activities has to determine the cost of doing so. Inaccurate product costing results in erroneous profit reports and misrepresents the profitability of products and manufacturing resources to the management of the firm. In addition, the impact on profit can not be assessed, which

means that management decisions will be based on incomplete information and may not take advantage of fluxuating costs appropriately.

F. SURVEY OF AUTOMATED SUPPORT FOR MANUFACTURING

The best motivation for our work can be found by surveying previous research projects and providing an analysis of them. These research projects can be divided into three categories. The first, *function-specific*, consists of database work which supports one of the basic manufacturing activities described previously. The second, *application-specific*, consists of database work which supports particular application areas such as VLSI design. Projects in the third category assume that the database support for manufacturing is already adequate.

1. Function-Specific Support

a. Support for CAD

The manufacturing function which has been the subject of the majority of studies concerning the application of database technology is computer aided design. As automation of the design function increases, emphasis is shifting from use of computers as straight numerical computing devices to the definition, manipulation, and enforcement of complex relationships among design objects [Ref. 28]. The potential for computers to assist engineers in performing design functions is placing a new requirement on data management systems to do more than store and retrieve ordinary textual data.

One aspect of the design function which has been studied is the concept of combining a set of engineering constraints with a database of engineering data [Ref. 29]. These constraints deal with the *semantics* of data and therefore define the limitations on the values that the data can take on. The ability to enforce these constraints determines a databases's integrity [Ref. 30]. Integrity checking has traditionally been performed by application programs, not by the data base manager. To ensure the correctness of design data, constraint management capabilities are being incorporated into engineering design database management systems [Refs. 31, 32].

Another extension to the traditional database management systems allows for the support of abstract data types to permit the use of new data types such as polygons, rectangles, and text strings [Ref. 33]. Current database management systems support the use of integers, floating point numbers, and character strings, all of which are widely used in business data processing applications. To properly model the semantics of design objects, new abstract data types are necessary.

An alternative approach to modeling the semantics of design objects was proposed which treat the features of the design object as primitives [Ref. 34]. The features which a designer uses are determined by the application domain he is working in. For example, when designing a casting, knowledge about features such as slabs and holes are useful since these features logically symbolize casting applications in terms that the designer understands. In this *designing-with-features* [Ref. 35] approach, a features database is used to store information about the features and their relationships. Standard operators exist to manipulate these features and relationships to build more complex design objects.

The *Integrated Programs for Aerospace-Vehicle Design (IPAD)* project is yet another approach to the limited data types inherent in the traditional database management systems [Ref. 36]. The IPAD project designed and developed a geometry data manager with special software driver routines which make the geometric objects available to application programs. This project is one example of many available where manufacturing companies have developed in-house systems to support manufacturing functions.

[Ref. 37] and [Ref. 38] introduce the notion of *complex objects* as an extension to the traditional relational system. The complex object crosses relation boundaries and groups related tuples from any number of relations and forms a hierarchy with a *root tuple* that defines the object. The defined objects are manipulated by the SQL language using minor extensions. Tuples in the database system are divided into two parts. The first part contains the data normally found in the data base - the data which is of concern to the designer. The second part of a tuple contains pointer information that is

used to link tuples belonging to the same object, information which is of no concern to the designer.

A similar extension to the traditional relational model allowed the integration of a constructive solid geometry [Ref. 8] scheme with a relational database management system. The SEQUEL query language was used by this particular system and required significant augmentation to support it. This approach has additional overhead in that the constructive solid geometry grammar must be converted into a generic data scheme which can be manipulated by SEQUEL. [Ref. 8] describes this conversion process as "computationally tedious".

b. Standalone Support for Other Functions

The Ford Motor Company initiated a project to develop a manufacturing database in response to their Basic Manufacturing Division's need for an automated system to support retrieval of process plans and detailed tooling and machining operations [Ref. 39]. This project used a relational database management system as the focal point of their manufacturing information system. The role of the database management system in this project was to replace a similar system in which manufacturing data was handled manually. The limited scope of application of the database permitted usage of the traditional relational system without any extension or enhancement.

The potential for database applications in flexible manufacturing systems is greater than for possible applications in standard manufacturing functions because of the highly automated nature of the machines and transportation systems employed in a typical FMS cell. In [Ref. 40], the layer concept used within various areas of computer science, such as operating systems, is adapted to FMS. Classes of objects are formed with respect to their common properties and are stored using a relational database system with a scheme wherein one relation represents one object class.

Process planning has been defined to be "situated at the information crossroads between product design and the shop floor" [Ref. 41]. The use of computers in process planning is natural because of their ability to make the numerous comparisons

necessary to formulate the best process plan in an efficient manner. Numerous projects have been initiated which apply database technology to the process planning problem. Lockheed-Georgia's Genplan system [Ref. 41] is a generative process planning system which has the logic and rules of manufacturing built into it. Genplan uses a relational database system to capture the data about manufacturing entities and their relationships and intends to use a knowledge-base management system [Ref. 42] to keep that data current. Several other projects have taken the same knowledge-based approach to process planning [Ref. 43,44].

Database management systems have also been applied to non-conventional machining processes to store machinability data [Ref. 45]. This machinability data is used to select metal cutting parameters based on the machining process to be performed and other major criteria such as accuracy, surface finish, power consumption, or economy. Although machining data handbooks satisfy most of the requirements for conventional machining processes, the automated systems support non-conventional processes and optimize the selection of parameters, something that the handbooks cannot do.

2. Application-Specific Support

One of the earliest efforts to develop an advanced data management system to support advanced applications concentrated on the implementation of a CAD database for the VLSI design environment [Refs. 46, 47, 48]. VLSI design was chosen because the products in that environment are typically large, complex, with components interconnected in a potentially complex manner, and therefore could not be handled by the available database models. Although some of the earlier work used the relational model in the underlying database system, it was recognized that this model doesn't sufficiently capture the relationships between different relations, a fundamental shortcoming of the relational model [Ref. 49]. Later efforts sought to overcome this problem by using an extensible object-oriented framework to model the VLSI design environment [Ref. 50].

Other application-specific projects have emerged in the areas of welding [Ref. 51], metrology [Refs. 52, 53], and chemical process plant design [Ref. 54], to name a few. Weldselector [Ref. 51] used an expert system approach to advise welding engineers on the selection of materials used to join metals, an ordinarily complicated task which is affected by factors such as the chemical and physical composition of the base metals, the position of the weld, and the degree and character of atmospheric contamination. The Weldselector program is a front end to a complex data base of information on a wide range of base metals, e.g., over 900 varieties of steel alone. In [Ref. 52] and [Ref. 53], research on the *Automated Manufacturing Research Facility (AMRF)* project at the National Bureau of Standards is presented. A major goal of this research is to develop a small batch manufacturing system to support research and experimentation in automated metrology (the science of measurement) and interface standards for the factory of the future. The major objective of the chemical process plant design project [Ref. 54] was to specify an overall systems architecture which truly reflected engineering design practice (the concentration of CAD work at the time was on VLSI design). This architecture was composed of individual databases to support project-wide applications, work area applications, and other smaller support and control applications. The research report was only a general overview of the project -- little, if any implementation work had been done.

3. Other Automated Support

The research projects included in this category include work on communicating manufacturing data using local area networks [Ref. 55], defining exchange formats and interface standards for communicating manufacturing data [Refs. 56, 57, 58], plant-wide computer control [Refs. 59, 60], use of engineering databases for decision-making [Ref. 61], and the use of browsing techniques in manufacturing databases [Ref. 62].

The work on local area networks proposed using distributed computer systems to place processing power where it is needed in the factory. The term "distributed" is in contrast to the "host type" architectures [Ref. 55], which are centralized computer systems supporting the entire spectrum of manufacturing functions.

Several exchange formats and interface standards have been proposed for communicating manufacturing data between and within design and production functions. [Ref. 56] discusses the requirements for such standards but acknowledges that: "The current status of product data communication efforts shows the need for more efforts for enhancing the interface specifications and turning them into standards." One such standard is the *Manufacturing Automation Protocol (MAP)* [Refs. 57, 58], a network architecture whose goal is to operate in a setting where products could be processed, assembled, and prepared for shipping without human intervention. The cost of installing MAP will limit its use to the automobile and aerospace industries [Ref. 57].

The use of computers to achieve plant-wide computer control is promising due to the increasing use of automation in factories. To reach this goal, disparate processors from different vendors must be linked to control both continuous and discrete manufacturing processes simultaneously [Ref. 59].

> Since the plant-wide computer control system will always be distributed to some degree because of the workings of the principles of locality and of autonomy, a good communications system will always be the heart of the overall control computer network. Therefore, a major disappointment to this author in reviewing the current progress and trends in the development of the new distributed control systems of the process control systems vendors, and indeed of the whole of the process control communications field in general, is the very wide variances in design and the consequent lack of standardization in their intra-system data highway offerings with all computer control systems but particularly with the new product line offerings now available. [Ref. 60].

G. SUMMARY

We have examined the role that automation plays in the operations of a manufacturing company by describing the major activities that make up the product life cycle. We have surveyed some of the current automation support available for these various activities. Our objective in this chapter was to introduce the state-of-the-art concepts and terminology used in the manufacturing environment, many of which will be used later in this dissertation.

III. COMPUTER INTEGRATED MANUFACTURING

A. BACKGROUND

The manufacturing industry is heavily influenced by pressure from the marketplace to reduce product prices while simultaneously increasing quality and responsiveness to customer demands. Many manufacturing companies, unable to keep the pace of change, are finding it more and more difficult to remain competitive. Solutions have been offered by manufacturing equipment vendors, computer manufacturers, and consultants to the problems of dealing with the broad, pervasive changes which are necessary for survival.

Manufacturers have turned to the computer, because of its widespread applicability, as a means of easing the pressure. Most of the applications to date have been aimed at specific manufacturing functions, such as engineering design, process planning, numerical control, etc. The increased use of computers in these specific areas has, in general, produced lower prices and productivity increases. However, in most cases, the actual benefits realized have been significantly less than expected. The application of computers in these specialized areas forms *islands of automation* which have contributed formidable problems in the attempt to produce further gains.

The major problem has been that the automated machines, control devices, and computers which form the islands of automation are acquired over time from different vendors and are unable to communicate or exchange data with other systems [Ref. 2]. This has been a major concern because the existing investments by manufacturing companies dictate that the multivendor, heterogeneous environment cannot be replaced in a wholesale manner, except perhaps in a few cases. The inability of these systems to communicate with one another has severely fragmented the information flow among manufacturing functions. The solution to restoring communications and information flow involves the integration of manufacturing functions using the Computer Integrated

Manufacturing (CIM) concept. Using this concept, links are formed between existing islands of automation, gradually evolving towards a totally integrated system.

Manufacturing companies want to link these islands of automation together into a system that still exhibits special characteristics, allows local control, and achieves high performance. They also want to have consistent data and control over larger portions of their operations, which is typical of centralized systems. The variety and complexity of the islands of automation dictate that the only workable strategy for CIM is a modular approach to integration.

B. WHAT IS COMPUTER INTEGRATED MANUFACTURING?

There is a wide diversity of definitions of CIM in the literature. Included are:

(1) "A collection of machines tied together by a material handling systems and controlled by a single computer or hierarchy of computers." [Ref. 63];

(2) "A production facility that consists of a group of process equipment units such as machine tools, auxiliary equipment (inspection machines, washing stations, etc.), linked with an automatic materials handling system that reaches every process station, the entire facility being integrated under common computer control." [Ref. 64];

(3) "CIM is the integration of key product-related data in a company, where the integration of various computer-based automation activities leads to improved productivity in all business areas from marketing to product shipment." [Ref. 65];

(4) "CIM is the vehicle that links the operations of the entire company together which results in a cohesive system." [Ref. 66];

(5) "CIM is a rounded concept that rests on a central manufacturing database. Linked to this database will be the key functions of engineering design, manufacturing engineering, factory production, and information management." [Ref. 67];

(6) "Computer integrated manufacturing is the automation and integration of the business of manufacturing from product design to distribution." [Ref. 68].

The first two definitions imply that CIM applies to a given set of machines, such as would be found in a FMS cell. They also emphasize the material handling system and common computer control. Both of these definitions limit themselves to a subset of the factory floor. The third and fourth definitions broaden the scope of CIM to include business and other non-engineering/production functions. The fifth definition adds the concept of a centralized database system and the sixth stresses the role of automation as an essential element of CIM.

Webster's Dictionary defines integrated as unified or united. We maintain, therefore, that the "integrated" in Computer Integrated Manufacturing should refer to the unification of the processes in the factory through automation of the data interactions between these processes. Our use of the term Computer Integrated Manufacturing or CIM uses the word manufacturing in the broadest sense to mean the use of automation to support all product life cycle activities, not just those concerned with the production phase of that cycle.

No matter which definition is used, CIM has several objectives which are provided for in all of the above definitions. One objective is to remove human intervention which normally results in improved quality [Ref. 1]. Another objective is to manufacture products in a flexible manner at minimum cost. Minimizing interruptions in the production process is sometimes yet another objective. Simply stated, the goal is to complete the production of an item in the simplest and most timely way, which will happen when each process flows automatically into the next, without interruption. Once CIM is implemented, the benefits will include real-time, on-line access to all data by the people and processes which need that data, higher quality, shorter design/production cycle time, efficient production of small batches, and faster incorporation of design changes into the system. All of these benefits mean better response to market demand for flexibility, quality, and fast delivery at the least cost.

C. CURRENT APPROACHES TO INTEGRATION

Computer Integrated Manufacturing uses automation to achieve integration in a manufacturing enterprise. The ideal CIM system would truly integrate Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), a Flexible Manufacturing System (FMS), and business data processing functions. Currently, each of these components is highly specialized and automated, but not fully integrated [Ref. 1]. We will compare three alternative approaches to achieving full integration using database technology. *High-level* integration interfaces one component of CIM (CAD, CAM, FMS, or business) to another by automating the data interface between the two components. Integration by *centralized database support* uses a database management system as the hub of the overall manufacturing system so that the data which is output from a given function is available to any other function that requires it. *Low-level* integration standardizes the data interactions between manufacturing functions using a distributed database management system so that one function can access data produced by another function.

1. High-Level Integration

The first approach we will discuss can be described as the high-level interfacing of the four main components of a CIM, i.e., CAD, CAM, FMS, and a business data processing system (see Figure 5). The primary motivation behind trying to provide for integration in this manner is to utilize as much of the existing automation investment as possible. In addition, given the amount of time which a CIM would take to implement, this may be a "quick fix". While this approach includes the most desirable form of coupling, data coupling, which occurs when all data required by one function is explicitly passed by another function, the transformations required to provide that coupling are costly in terms of execution time. These transformations are implemented by translators which take output data from one component and convert it into the form necessary for use by another component. The translators would operate in one direction only and would not provide the degree of interactivity normally required for effective decision-making. Bidirectional communication between two components would require two translators. In

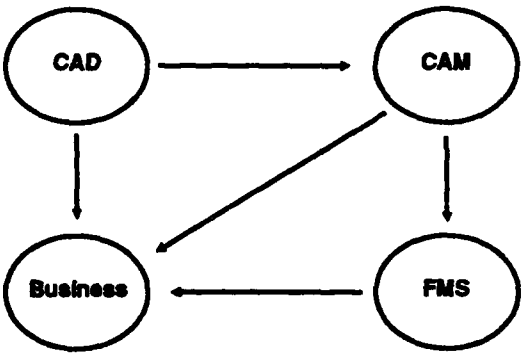


Figure 5. High-level Integration

this high-level approach, the workings of the low-level functions within a component are hidden from the other components and their low-level functions.

Several research projects have taken this approach. [Ref. 69] proposed linking islands of automation using three separate interfaces. The first interface, between material requirements planning and CAD, would be used to communicate bill of material data from CAD to MRP. The second links MRP and the automated storage and retrieval system to automatically transfer pick requests from MRP to the retrieval system, in lieu of manual communication. The third link, between MRP and automatic test equipment, serves to feed measurement data generated by the test equipment to the MRP system for quality control functions.

The Hewlett-Packard Company produced a system called DesignCenter [Ref. 70] which provided *design acceleration tools* and links between various design functions. One link was established between software design and hardware logic design and simulation (CAE). A second link was used between CAE and CAD, and the third was used between CAM and the board testing function. Although this system served mainly design functions, the approach used to implement the links is analogous to the high-level interface approach.

[Ref. 71] discusses the link between JIT and CIM and the impact that correct scheduling rules have on the JIT philosophy. As a premise to this discussion, the concept of high-level integration of CAD and CAM is introduced. A similar approach to the

integration of CAD and CAM is taken in [Ref. 72], which proposes the use of software translators to communicate among CAD and CAM databases.

This approach to integration should be viewed as a short-term solution. Even though some increases in productivity and efficiency may be realized, it still has the disadvantages associated with solving localized problems, the major problem being that it doesn't integrate the functions in the context of the entire manufacturing process.

2. Integration by Centralized Database Support

An alternative to the previous approach is to integrate the four main CIM components using a centralized database (see Figure 6). This alternative is generally the approach taken by the process-oriented integration proponents because of the ease of query processing and performance of database administration functions. This alternative is normally unrealistic due to the heterogeneity of the functions to be supported [Ref. 16]. In addition, concurrency control is complicated due to the need for prioritization of real-time access requirements. In a sense, this alternative provides too much integration. Corporate planners, who normally are interested in the business data processing component, e.g., summary information about shop floor productivity, have no interest in data such as the maintenance status of a machine on the shop floor, even though the information is readily available.

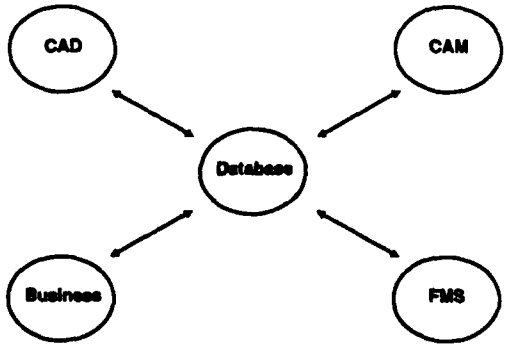


Figure 6. Centralized Database Support

The Boeing Commercial Airplane Company proposed an approach which used a *geometry engine* as the nucleus of the system [Ref. 73]. This geometry engine consists of a user interface, a data modeler, a local data manager, and a communications processor. Surrounding this nucleus are applications modules performing such functions as tool design, production drawing, and numerical control programming. Each module is independent of the nucleus, but uses the same data modeler, data manager, and communications processor. The output of the CAD/CAM geometry engine will go into a common data management system.

[Ref. 74] supports the idea that "The most essential part of any CIM system is the common data base that includes both geometric and non-geometric product information." The reason that the data base system is so important is because "the greatest productivity gains and largest cost savings can be achieved only through the development and use of a common geometric data base for design, analysis, drafting and production." The centralized approach eliminates the lengthy process of re-creating basic design data and avoids the errors due to transcription of this data from one system to another.

Another benefit to the centralized approach is that communication between design, production, and their various functions and activities is improved as many of the traditional barriers are broken down. Two fundamental aspects of the design and manufacturing process benefit from this centralized approach. First, the entire manufacturing process, from product design to service support is a *monolithic*, indivisible function [Ref. 75]. The interrelationships of all the various components dictate that no single portion can be considered on its own, but must be considered in the context of the entire process. Second, the common ingredient in all manufacturing operations is the data which is created, stored, analyzed, transmitted, and modified.

The *Computer Assisted Document Management and Control (CADMAC)* system [Ref. 76] is one example of the communication of data using a centralized system. This system stores both computer generated CIM files and raster images of paper documents in digital form. Once these files are stored in the centralized database, they can be cataloged, located, retrieved, edited, printed, and distributed electronically, which

improves productivity and product quality. The true power of the CADMAC system is that users can access required documents to answer queries for information about those documents. This capability results in significant time savings by avoiding the manual searching through files for this information.

The *CODASYL* database system [Ref. 30] has been used as a centralized database management system supporting CIM in the metalworking industry [Ref. 77]. Because of the difficulty of navigating through a network database, a more "user friendly" interface was developed to simplify the work of the applications programmers and the engineers using the system.

There are many reasons why the centralized database approach is not an acceptable alternative in providing integration for manufacturing functions [Ref. 16]. Such a centralized system would be responsible for monitoring real-time manufacturing processes, maintenance of all aspects of the database including static data, e.g., initial setpoints, alarm limits, engineering unit conversions, etc., and dynamic data, e.g., current point values, current alarm state, etc., handling operator access to the system, and a whole variety of other tasks. Few, if any, computers are currently available which could handle all of these functions in a timely manner [Ref. 55]. Even if such a computer exists, the use of a single centralized computer poses other problems. A major difficulty with the use of a single system is the vulnerability to system failure. When all data flows through a single host, the entire system ceases to operate when that host fails. Some portions of the factory will still have enough autonomy to be able to continue operating in a standalone mode, but from an overall control point of view, the system is inoperative. This problem can be minimized by providing backup computers, but the benefits rarely warrant the expenditure. Most companies would be tempted to adopt an optimistic philosophy and disregard the need for backup.

Another problem with a centralized system, already mentioned above, is the limited capacity that a single machine would have for handling the massive data communications requirements to support an average sized factory operation. Similarly, the storage and manipulation of this massive volume of data by a single database

management system would be impossible. The volume of data required in the design function alone would consume the capacity of most mainframe computers.

If a centralized approach were taken, the diversity of data to be supported would result in a low degree of semantic expressiveness within the database management system. The additional semantics required to properly model the manufacturing environment would have to be provided by application programs which interface to the database. In this case, which is analogous to traditional data processing, the database management system is reduced to being a file server for those application programs.

> Many people think that CIM means putting all of a corporation's data on a data base management system (DBMS). This is neither a desirable nor achievable goal. The great revolution in mini- and micro-computers was largely fueled by the poor performance of large shared systems. Even logical centralization of data is a spurious goal for all data of an enterprise. One should not expect that corporate planners would be interested in stresses on a part or that an engineer would be interested in the maintenance status of a machine on the shop floor. Rather, data should be organized so that people or machines that share a set of functions have access to them. A centralized data base ignores the heterogeneity of data management strategies and tools used in manufacturing today. [Ref. 16]

3. Low-Level Integration

The third approach to integration uses a distributed approach to organize the data, where each low-level function has its own database (see Figure 7). By standardizing the data interface between functions and databases, a function can access the databases of other functions since the data access protocol is the same between a function and any database. This, of course, can be achieved by requiring the databases to support a uniform data model and language. The question is, which data model and language is powerful and flexible enough to support various different semantics, or abstraction concepts, which are inherent in the various manufacturing activities? In other words, is there a single data model that can capture the data requirements of design, process planning, scheduling, group technology, etc.?

This standardization would force manufacturing system vendors to provide an interface with each new product to be used in the manufacturing process. In addition, standardization would allow application vendors to depend on the data in the system

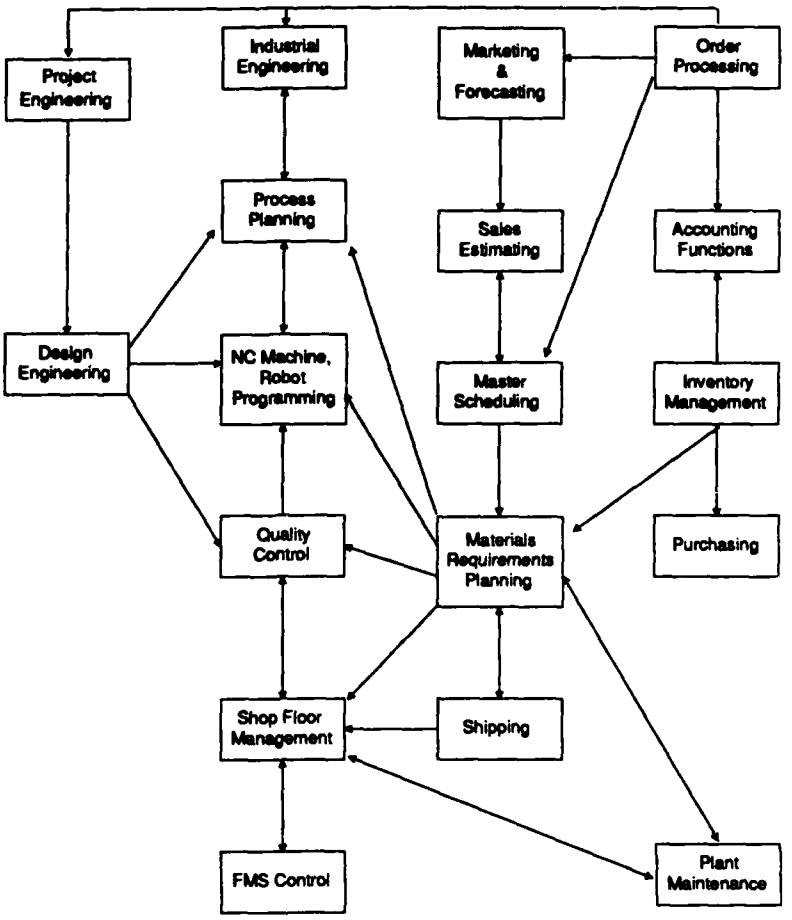


Figure 7. Low-level Integration

being in the proper form for interface to their products, facilitating development, installation, maintenance, and flexibility of the user to choose among competing products. This approach also has the normal advantages of a distributed system, including the flexibility gained by distributing the data, and the normal disadvantages of a distributed system, including the inefficiency of querying data distributed over several sites.

[Ref. 78] proposes a distributed approach to CIM, but does not give a definitive architecture to support it. There is no indication of how the databases should be distributed or what functions they would support.

This approach is also considered a long-term solution. Again, integration has to be planned in advance to produce the best results. This approach is advantageous over the centralized approach because it alleviates the problems associated with a single centralized database.

D. SUMMARY

In this chapter we discussed the manufacturing industry's proposed solution to the islands of automation problem, a concept called Computer Integrated Manufacturing (CIM). The wide diversity of definitions of CIM leads us to believe that there is no industry-wide consensus about the definition of the concept. We therefore, adopted a definition which is consistent with our data-oriented approach. We have also examined three alternative approaches to integrating manufacturing functions. Of the three, we feel the low-level approach has the most potential for the long term. The cost of implementing this low-level approach may make it unrealistic for small and medium-size companies that cannot afford to ignore their current investment in computers and manufacturing equipment. The best solution for these companies seems to be the high-level integration approach we have described. We agree with other researchers that the centralized approach is not viable and therefore we have not pursued it further.

IV. DATA MODELING

A. BACKGROUND

> It is apparent that an interpretation of the world is needed which is sufficiently abstract to allow minor perturbations, yet is sufficiently powerful to give some understanding concerning how data about the world are related. An intellectual tool that provides such an interpretation will be loosely referred to as a *data model*. It is a model about data by which a reasonable interpretation of the data can be obtained. A data model is an abstraction device that allows us to see the forest (information content of the data) as opposed to the trees (individual values of data). [Ref. 79]

To better understand data modeling, it is helpful to define what the objects are that are being modeled. [Ref. 80] proposes the tuple

< object name, object property, property value, time >

as a working definition of an atomic piece of data. This tuple represents an object (*object name*) and some aspect of that object (*object property*) which is captured by a value (*property value*) at some point in time (*time*). The modeling of time is covered in [Ref. 81] and is beyond the scope of this work. Several data models have been developed which represent and relate an object name, object property, and property value. One way of relating data is to categorize them according to their properties [Ref. 82]. In a given data model, the names of the categories together with their properties is called a *schema*. The schema also includes relationship information for the categories and properties. Figure 8 gives an example of a schema with three categories, **employee**, **firm**, and **car**. The categories are depicted by ovals, properties by rectangles, and relationships by lines between the categories they relate to.

A data model defines the rules according to which data are structured and the operations which can be performed on the data being represented. A structure can be as simple as a list of objects which can represent a stack or queue, depending on how the operations are defined to operate on the list. The allowable structures for data within a data model are *static* in nature, that is, they are relatively time-invariant, and are normally

defined by a *data definition language (DDL).* The operations defined for the data model are dynamic in nature since they cause a change in the various values that the data take on. These operations normally comprise the *data manipulation language* for the data model.

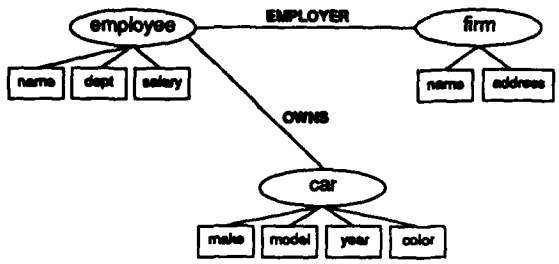


Figure 8. Database Schema

The combination of structure and allowable operations determines a unique data model. Given the number of possible alternatives, many different data models could be specified. Practicality and usefulness limit the number of data models which have actually been used. Three of these, the hierarchical, network, and relational models, are the most widely accepted and used. These three models will be discussed in more detail and will be hereafter referred to collectively as the *traditional models*.

B. TRADITIONAL DATA MODELS

1. Hierarchical

Historically, hierarchical systems are the oldest of the database systems in use, and the hierarchical data model is the oldest of the traditional data models [Ref. 83]. The structure of a hierarchical data model appears to the user as trees of interconnected segments (see Figure 9) where the relative order of the subtrees is important. The arcs connecting nodes in the tree always point toward the leaves and away from the root. The diagram in Figure 9 represents an *intension* [Ref. 79] of a hierarchical database. This

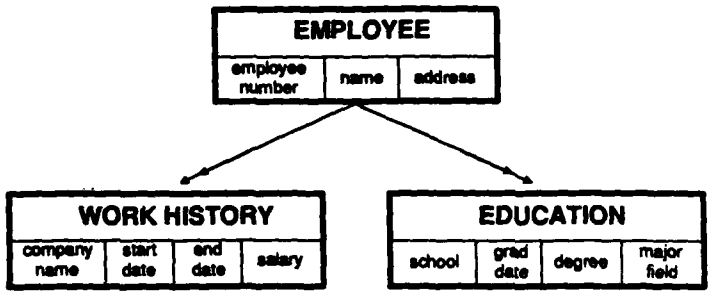


Figure 9. Intension of a Hierarchical Database

intension describes the structure of the database in terms of its *segment types* and the relationships between them.

In this figure, **employee**, **work history**, and **education** are segments. Each segment is composed of one or more *fields*. The relationship between the **employee** segment and the **work history** segment is a *one-to-many relationship* [Ref. 79], that is, there may be more than one occurrence of work history data for a particular employee. The same type of relationship exists between the **employee** segment and the **education** segment. The one-to-many relationships are represented by the double arrows in the intension diagram. Relationships in a hierarchical data model are also called *parent-child relationships* [Ref. 83]. In the example above, **employee** is the parent of both **work history** and **education** (the children). **Work history** and **education** are related as *siblings* [Ref. 83].

Figure 10 shows a *record* which is an *extension* of the structure shown in Figure 9. An extension of a segment is a group of data items relating to a specific entity.

While the simplicity of the hierarchical model seems attractive, it does have some limitations. The model only permits representation of one-to-one and one-to-many relationships directly. Many-to-many relationships require an artificial segment to be inserted as shown in Figure 11. Here, **parts** has a many-to-many relationship with **supplier**.

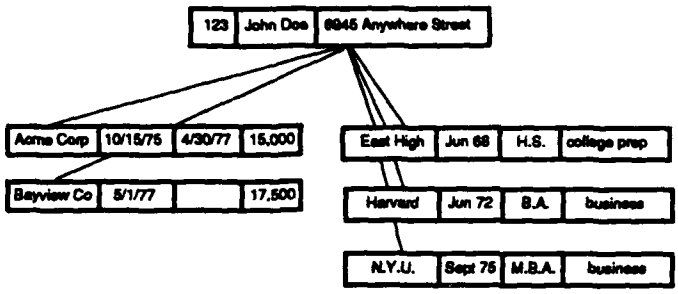


Figure 10. Extension of a Hierarchical Database

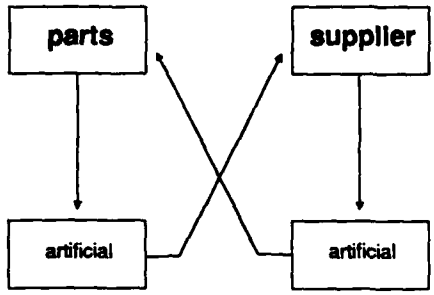


Figure 11. Many-to-Many Relationships

The only type of relationship which can be modeled in the hierarchical model are *binary* relationships between two segments. If two segments are related, only one relationship can exist between them. One segment must serve as the *root* to maintain the tree structure.

2. Network

The most prominent network data model was developed by the Data Base Task Group (DBTG) of the Conference on Data Systems Languages (CODASYL) and is known as the *CODASYL network data model* [Ref. 84]. In this model, entities are represented by *records* which are groups of related fields. The relationships between entities are represented by *sets* among the record types. Each set has a designated *owner*

record type and may contain one or more record types as *members*. Figure 12 depicts a sample application modeled as a network.

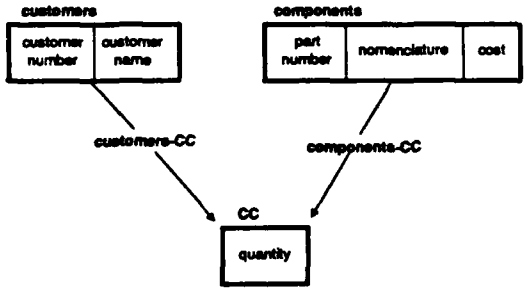


Figure 12. Intension of a Network Database

This example contains three record types, **customers**, **components**, and CC. **Customers-CC** and **components-CC** are the sets which relate **customers** and **components**, respectively, to CC. Each occurrence of **customers-CC** consists of a single occurrence of **customers** (the owner) and one occurrence of CC (the members) for each order in which that customer appears. Likewise, each occurrence of **components-CC** consists of a single occurrence of **components** (the owner) and one occurrence of CC (the members) for each order in which that component occurs. Figure 13 provides sample data values for this network.

Again, as with the hierarchical model, only binary relationships which are one-to-one or one-to-many are directly represented in the network data model.

3. Relational

The relational data model is rapidly becoming the most popular of the traditional models. It differs in several aspects from both the hierarchical and network models. First, the relational model is based on a theoretical foundation from relational mathematics. Second, the relational model is more abstract than the other traditional models. The relational model represents data in a more natural way - closer to the way the data exists. The hierarchical model requires data to be represented by hierarchical

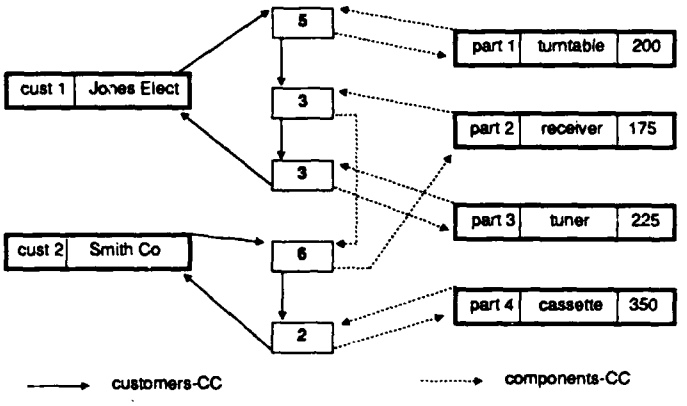


Figure 13. Extension of a Network Database

constructs, whether that type of construct is appropriate or not, and similarly, the network model requires the use of the set concept. Instead of forcing an artificial construct on the user, the relational model reduces relationships to simpler components and then represents those components directly. The major reason for the increasing popularity of the relational model is that it tends to simplify rather than complicate the user's view of the data.

The relational model is built around the concept of a *relation*. A *mathematical relation* is a set that expresses a correspondence between two or more sets, S_1, S_2, ..., S_n [Ref. 79]. An *n*-ary relation T is defined as a subset of the Cartesian product of its domains ($T \subseteq S_1 \times S_2 \times ... \times S_n$).

The mathematical concept of a relation is used in database theory to define a *database relation*. A *relation scheme* **R** is a finite set of attribute names $\{A_1, A_2, ..., A_n\}$. Corresponding to each attribute name A_i is a set D_i, $1 \leq i \leq n$, called the *domain* of A_i. Let $\mathbf{D} = D1 \cup D2 \cup ... \cup D_n$. A *relation* **r** on relation scheme **R** is a finite set of mappings $\{t_1, t_2, ..., t_p\}$ from **R** to **D** with the restriction that for each mapping $t \in r$, $t(A_i)$ must be in D_i, $1 \leq i \leq n$ [Ref. 85].

A relation scheme appears to the user as a two-dimensional table of data whose entries are atomic values. In the standard relational model, no repeating groups or other

complex structures are permitted as entries in the relation. In addition, all of the entries in any one column are from the same domain. The columns of a relation are called *attributes* and have unique names. The order of the attributes in the relation scheme is immaterial. The rows of the relation, called *tuples*, can also appear in any order, and have the additional restriction that no two rows in the relation are identical.

Figure 14 gives an example of a relation called **STUDENT**. Note that this relation has four attributes, **student number**, **name**, **academic major**, and **advisor**, and therefore the tuples in this relation are called *four-tuples*. The domain of the attribute **student number** is the positive integers and the domains of the other attributes are characters of length 15, 10, and 10 for **name**, **academic major**, and **advisor**, respectively.

STUDENT

student number	name	academic major	advisor
1101	Joe Jackson	biology	Smith
1120	Sue Anderson	physics	Newton
1123	Rusty Springs	biology	Jones
1205	I. Want Moore	english	Glover

Figure 14. Relation

The tuples in a relation are identified by the values of its attributes. One way to identify a tuple is by listing the attribute values for every attribute in the relation. In the example above, **1120, Sue Anderson, physics, Newton** constitutes a unique identifier for the second tuple since no two rows in the relation can have identical attribute values for all attributes. It generally is possible to identify a tuple by specifying fewer attribute values. In the **STUDENT** relation, the attribute **student number** alone will uniquely identify a tuple since each student is assigned a unique student number. Each of the other attributes alone may not be sufficient to uniquely identify a tuple. Two students could have the same name and there will surely be a case where two or more students have the same academic major. The same holds for the advisor attribute. Any combination of one

or more attributes which uniquely identifies a tuple is referred to as a *candidate key* [Ref. 30]. In the relational model, one of the candidate keys in each relation is selected to be used as the tuple identifier and is called the *primary key*.

One major goal of the relational model is *data independence*. Data independence is a measure of a database system's ability to provide for change in representation or in content of the database without affecting programs [Ref. 83]. This is achieved by representing data as relations and deferring the definition of relationships among relations until execution time, when either *relational algebra* [Ref. 85] or *relational calculus* [Ref. 85] can be used to express the relationships using the values of common domains in the relations concerned. In theory, the data is represented logically and the operations on the data are represented logically. This is not true of either the hierarchical or network models. Furthermore, in the latter cases, the user can only process data using the hierarchical or set relationships defined by the respective hierarchical or network structure.

4. Limitations of the Traditional Models

The limitations of the traditional models are addressed in [Ref. 49] and [Ref. 83]. The limitations discussed here will focus on those most relevant to the manufacturing environment. It has been generally stated that the traditional models are not well-suited for manufacturing applications [Refs. 8, 29, 33, 36, 37, 38, 48, 50, 52, 54, 83, 86]. The two major objections cited are the lack of support for *abstract data types* [Refs. 33, 36, 37, 38, 86] and limited *semantic expressiveness* [Refs. 8, 48, 83, 86]. Of the two, the limited semantic expressiveness seems to be most serious drawback. Given the record-oriented nature of the traditional models, the mapping of application semantics into a low-level record-based structure tremendously limits their semantic modeling capabilities [Ref. 49]. The simple data structures used by the traditional models to model semantics often cause loss of information and therefore only support a limited portion of the application environment semantics [Ref. 87]. The basic problem with the traditional models is that they fail to distinguish the different kinds of relationships among the objects in the application environment. The same data structure describes the attributes of

an object, the type of that object, and the relationships between types, again, causing loss of information.

The lack of support for abstract data types results in complex objects from an application environment being represented by record structures, a correspondence which is unnatural and difficult for users to cope with. Users should be able to address and manipulate objects supported by a database system in the same way they are addressed and manipulated in the application environment, which is the major purpose in using a data model.

C. SEMANTIC DATA MODELS

1. Background

Semantic data models attempt to provide high-level data structuring features to improve the expressiveness of database conceptual schemas. This is done by embedding the semantics of a particular application in the database schema. The overall objective of the semantic models is to increase database accessibility by end users, many of whom are not trained in computer science.

In addition to providing for the representation of these semantics, the ideal CIM data model would provide other features which are not found in the traditional models. One of these features is the representation of design objects as primitives in the model, with prescribed "rules" for associating objects with one another. These objects could be the building blocks from which more complex objects could be built. Operations defined for the data model would include those for manipulating objects. These operations would include provisions for adding new objects and modifying existing ones.

Semantic data models are normally represented by a set of *abstraction concepts* which they employ to enhance their modeling capabilities. Many of these abstraction concepts are rooted in the area of artificial intelligence (AI) known as *knowledge representation*, in particular, *semantic networks*. The main difference between the work

done in AI and semantic data modeling is that the AI researchers are more concerned with representing abstract information rather than information structured in a manner oriented toward database applications [Ref. 83].

2. Abstraction Concepts

a. Generalization/Specialization

Generalization refers to abstraction in which a set of similar objects is regarded as a generic object [Ref. 88]. Generalization is used to classify objects into types, which can be classified into other more general types. The generalization abstraction concepts places the emphasis on the similarities of objects and abstracts away their differences [Ref. 79]. Figure 15 is an example of a generalization hierarchy for a subset of a data processing organization. The arrows in the figure indicate the direction of generalization. For example, **employee** is a generalization of **clerical**.

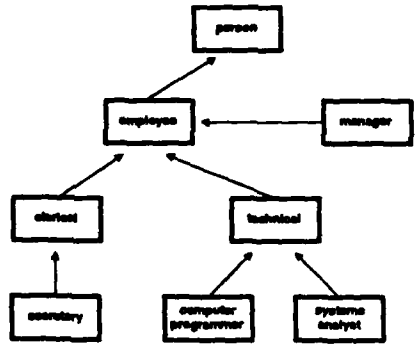


Figure 15. Generalization Hierarchy

One of the benefits of using generalization is that *inheritance* can be used between related types. In our example, all of the properties of **employee** can be inherited downward to both **clerical** and **technical**, whose properties in turn are inherited downward further in the hierarchy. If an **employee** is required by law to be over the age of 18, then inheritance will stipulate that anyone who is either **clerical** or **technical** must

be over 18 years of age as well. This downward inheritance will always produce valid results [Ref. 89].

Specialization is the opposite of generalization [Ref. 79]. In Figure 15, **employee** is a specialization of **person** (reverse the arrows in the generalization to obtain the direction for specialization). One important distinction between generalization and specialization is that specialization doesn't always allow for inheritance of properties in the way that generalization does. For example, if all **computer programmers** are paid less than $30,000 per year, it does not necessarily follow that all **technical** employees are paid less than $30,000 per year; a **systems analyst** could be paid $40,000 per year.

b. Aggregation

Molecular *aggregation* is the abstraction of a set of objects and their relationships into a higher-level object [Ref. 88]. This abstraction allows a view of objects from different levels of generality, each with its own level of detailed definition. A user interested in the overall design could use the topmost level of abstraction, which would hide the implementation details. This implements the *Information Hiding* [Ref. 90] principle commonly found in programming language design. The idea is to give the user only the amount of implementation detail he needs for a particular application. Figure 16 depicts **person** as a molecular aggregation of **name**, **address**, **age**, **date of birth**, and **birthplace**. All of these except **address** are primitive objects, i.e., they are not further divided. Note that two levels of molecular aggregation abstraction are present in

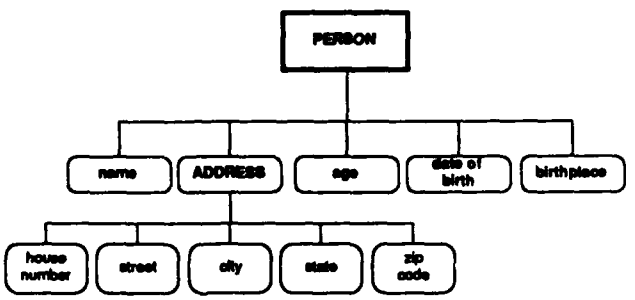


Figure 16. Aggregation

the figure. The objects whose name appears in upper case are molecular aggregations. Those in lower case represent primitive objects in this example.

The properties of a type, such as **name, address, age**, etc., are referred to as *intensional* properties [Ref. 91] (*intensions*) because they are definitional in nature. In fact, aggregation is the normal means by which we describe or define items, we specify the properties that the object takes on. The values that these properties can take on, such as John Jones, 123 Anywhere Street, etc., are *extensional* properties [Ref. 91] (*extensions*) since they are factual as opposed to definitional.

Molecular objects have two description components, an interface, and an implementation [Ref. 92]. The interface specifies the general function of the object and the implementation provides the details of the use of the object for a particular application. The aggregation concept will be discussed further in Chapter V.

c. Association

Association is a form of abstraction in which a relationship between similar objects is considered as a higher level set object [Ref. 93]. The relationship is regarded as a membership relation. The details of the member objects are suppressed and the properties of the set are emphasized. Figure 17 gives an example of a **country club** with an association of **golfers**. The properties of **golfers** are specified to be **name, address, handicap**, and **annual dues**.

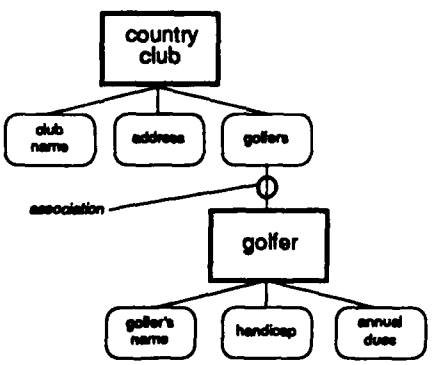


Figure 17. Association

d. Version Generalizaton

Version generalization is a form of abstraction in which similar objects are related to a higher level object, generally a type [Ref. 92]. A type is an abstraction of the common properties of its versions. This abstraction features inheritance which is analogous to that for the generalization abstraction concept. Versions can have two distinct forms of attributes; those shared with the object type, and those defined to be unique for each version. Attributes shared with the object type reproduce the interface characteristics of the object type. Attributes defined to be version specific are the attributes which distinguish one version of a particular type from another version of the same type. Figure 18 provides an example of a set of object types and a related set of object versions. **Car** and **truck** are types which are related according to the diagram to **convertible, station wagon, 4 wheel drive pickup, short bed** and **long bed**. In this example, **car** would be defined as having either a canvas top or a station wagon top. There would not be any notion of a standard sedan in this case.

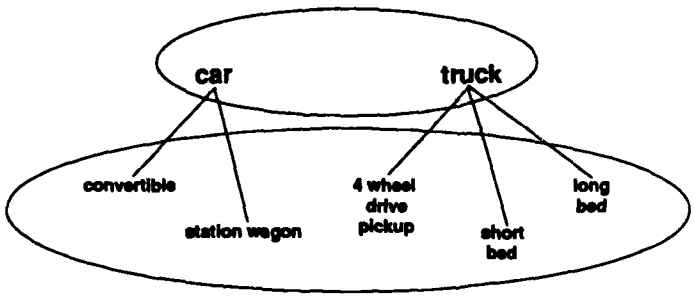


Figure 18. Version Generalization

Version generalization differs from the generalization concept defined previously in that version generalization specifies the relationship between an object type and its versions, while ordinary generalization is used to specify the relationship between a type and its subtypes.

e. Instantiation/Classification

An object is created by *instantiation* [Ref. 92]. Both object types and object versions can be instantiated. Creating multiple instances of the same type/subtype or version provides for a distinction between the various copies. A version may be instantiated to provide a local working copy of a previous design, which can be specified to any level of detail. Types (or subtypes) can be instantiated to produce a working copy for design work from scratch, in cases where no existing design can be used. Figure 19 shows an object **Fred's Car**, which is an instance of type **CAR**. **Fred's Car** would be produced to provide a working copy of type **Car** as a starting point in this particular design. The fact that **Fred's Car** is instantiated from its parent type tells us that the implementation specifications for the final product are not available and will be developed from scratch. If **Fred's Car** were instantiated from **Red Convertible** instead, the design would begin from the point in **Red Convertible** where implementation details left off, indicating that some similarity exists between the implementation of **Fred's Car** and **Red Convertible**.

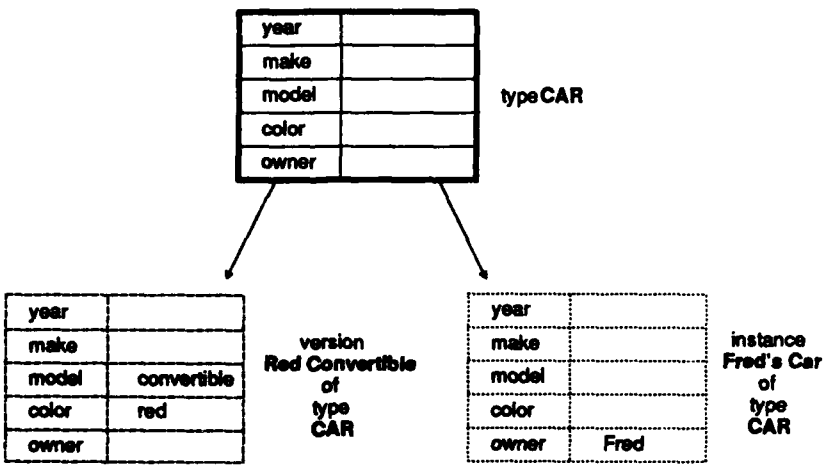


Figure 19. Instantiation

Classification, the opposite of instantiation, defines an object type as a set of instances [Ref. 79]. Each instance shares common characteristics with the other members of the same class. For example, in Figure 20, the instances **Joe's Truck** and **Sam's Truck** define the type **TRUCK** through classification.

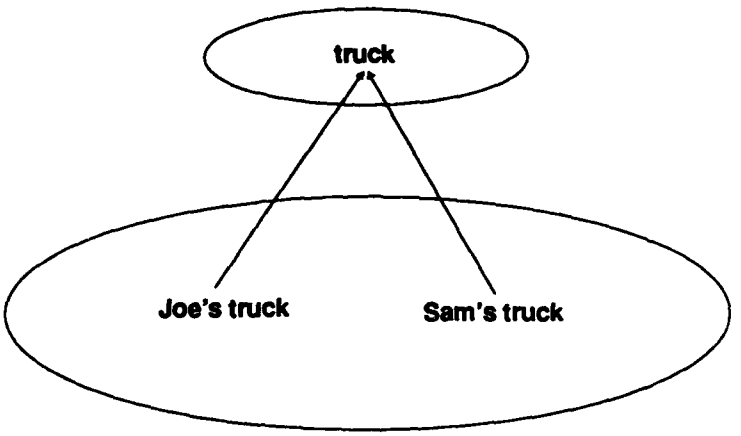


Figure 20. Classification

f. Version Hierarchy

A hierarchy is formed for the set of versions for a particular type/subtype, and is called a *version hierarchy* [Ref. 94]. In this hierarchy, going from a higher level to the next lower level, we find that more implementation details are specified. The difference between the type/subtype generalization and the version hierarchy is that different versions of an object have the same set of attributes, and not necessarily the same values, while different types (or subtypes) will have different sets of attributes from each other. Figure 21 depicts three version hierarchies. In this case, **RANCH** is a subtype of type **HOUSE**, and two bedrooms, three bedrooms, and four bedrooms are subtypes of ranch. Each subtype can have its own version hierarchy. The blocks labelled **10x12 Master, 12x15 Master,** and **14x18 Master** are on the same level in the diagram because they represent mutually exclusive versions. Each block in the diagram is a potential starting point for future designs.

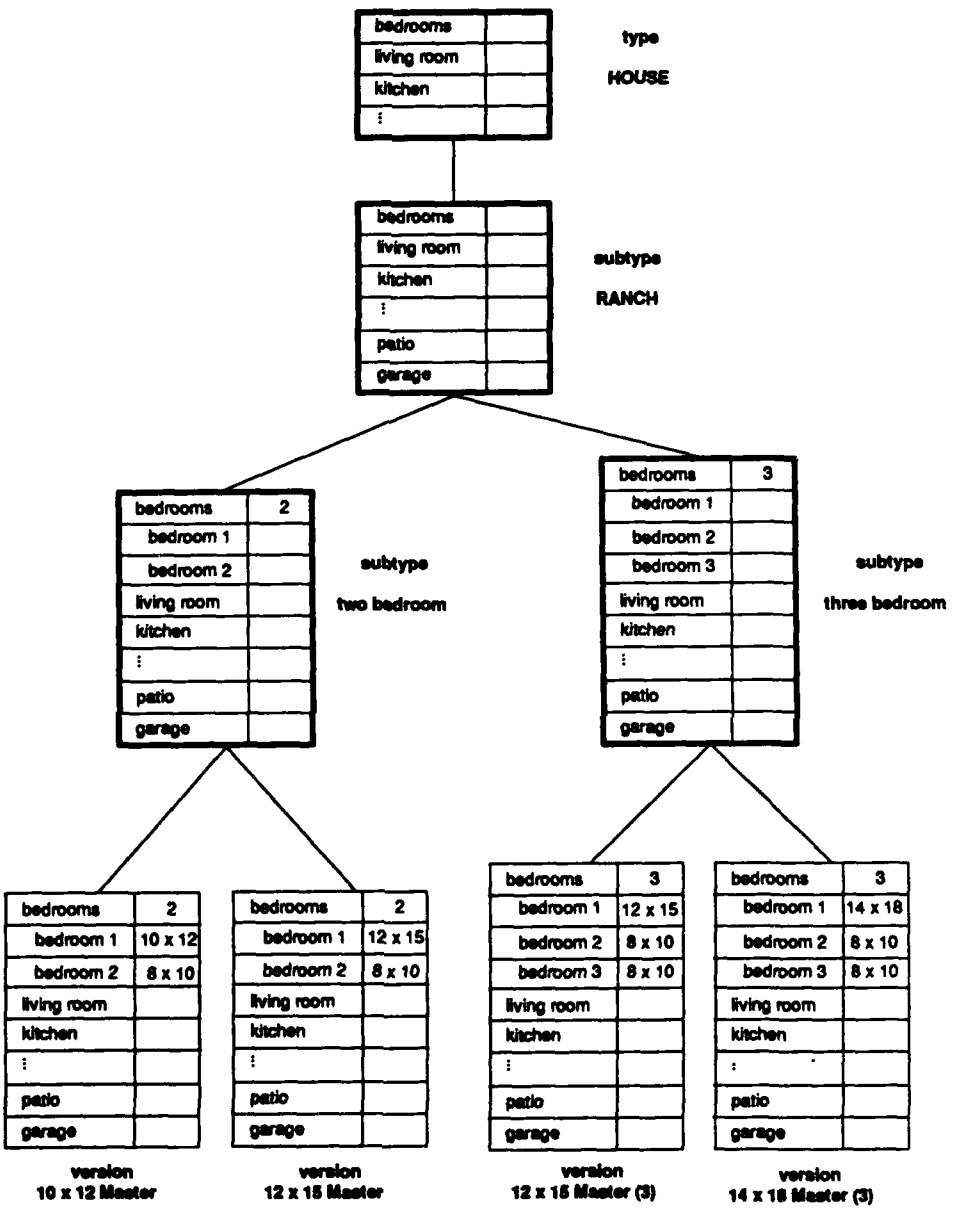


Figure 21. Version Hierarchy

g. Instance Hierarchy

The instantiation abstraction is extended to form an *instance hierarchy*, consisting of different instance alternatives for the same type/subtype or version [Ref. 94]. Figure 22 is an example of an instance hierarchy for a house being designed for John Jones. Since Mr Jones is building this house from scratch, the design starting point was an instantiation from subtype ranch. In the course of designing his house, Mr Jones wasn't sure whether he wanted his living room dimensions to be 15x21 or 17x19, two alternatives represented in the hierarchy. The reason for saving the hierarchy is that Mr Jones may decide on one size, finish the design, and then change his mind. The hierarchy would permit him to go back to the point of the decision and re-complete the design, which may require modification to other room dimensions. All of the information provided in the original design would be saved in the event he changed his mind again.

3. Survey of Current Semantic Models

Current semantic models include the Entity-Relationship (ER) Model, Functional Model, SHM+, SDM/Event Model, TAXIS, SAM*, and RM/T. All of these models use primitives such as entities, events, or simply objects. They also include provisions for composite objects and attribute specification among the supported features. Extended semantic models integrate a number of programming language concepts with database concepts. They also make use of advanced data type concepts such as abstract data types and strong typing. These extended models include SHM+, TAXIS, and the SDM/Event Model. Semantic modeling theory is now being applied to particular application areas such as office automation, VLSI, and cartography, as well as for traditional data processing applications (inventory, insurance). We will make use of many of the concepts from current semantic models in the description of our model.

a. Entity-Relationship Model

The *Entity-Relationship (E-R)* model [Ref. 95] uses a network representation to model objects (the entities) as nodes and relationships as edges between the appropriate nodes. This model identifies four levels of views of data which are used

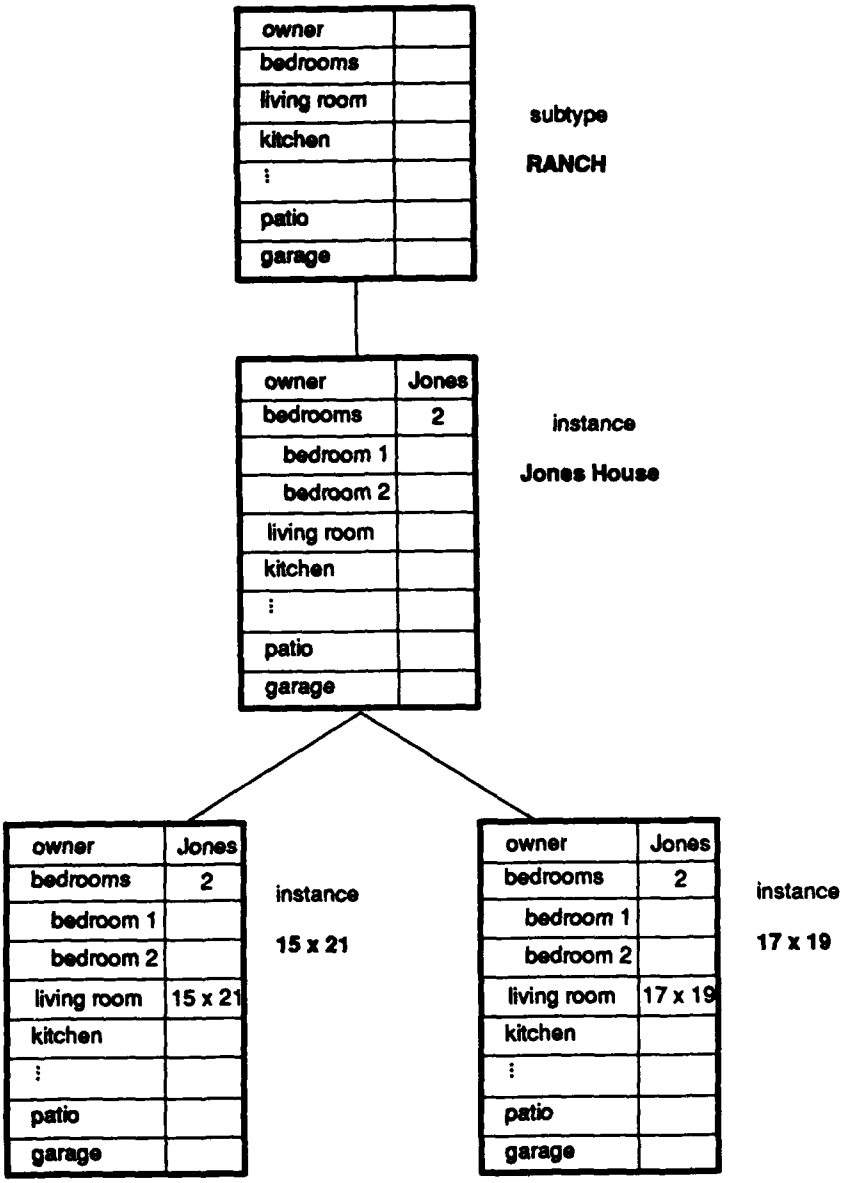


Figure 22. Instance Hierarchy

to support logical and physical database design. The first level, the semantic level, deals with information concerning the conceptual objects and relationships of interest. The second level organizes the information modeled in the first level into relations. The third level is concerned with the access-path-independent storage structures; those not involved with search or indexing schemes, which are placed at level four.

The E-R model supports many-to-many relationships using the notion of a *relationship set*, which is a mathematical relation among two or more entities. Both entities and relationships have associated attributes which define their properties. The main use of the E-R model has been in high-level database design [Ref. 79].

b. Functional Model

In the *functional database model* [Ref. 83], the attributes of an object are viewed as mappings from that object to some other domain of objects. One unique characteristic of this model is its integrated view of data definition and data manipulation. The traditional models separated these two activities into static and dynamic parts. Data definition, the static part, is done as part of the database design process. Once that definition is made to the DBMS, data can be entered, manipulated, and output using a data manipulation language. This is a dynamic activity; it depends on the state of the database, and changes the database from one state to another.

There are three predominant functional database models in existence. The *DAPLEX* [Ref. 96] model uses functions to define types and relationships among objects. Types are modeled as functions without arguments and relationships are modeled as functions with one or more arguments. Functions are manipulated using predicates and can be composed (as in mathematics) to form complex objects.

The *Functional Query Language* (FQL) [Ref. 97] models an application using a set of abstract data types and a set of functions defined on those types. In this model, which is mainly a query language, data manipulation using functions is done similarly to the way the DAPLEX model does it.

The *functional data model* (FDM) [Ref. 98] uses a graphical schema with nodes, which represent types, and mappings between the nodes. The functions can be many-to-one, one-to-one, or identity functions; can be either partial or total functions, onto or into, and may use ordinal types as domains. Data manipulation is done by retrieval and manipulation primitives which treat functions as logical access paths, and perform ordinary insert/update and delete operations.

c. Extended Semantic Hierarchy Model

The *extended semantic hierarchy model (SHM+)* [Ref. 99] extends the traditional relational model by providing more domains and data types for modeling complex relationships, makes a clearer distinction between the schema and database levels, and provides a constraint facility. SHM+ also employs the generalization and aggregation abstraction concepts to define type hierarchies and provide an inheritance mechanism. The subtypes in the hierarchies partition the instances of the parent type and may themselves be subtypes. With inheritance, some of the attributes of a subtype can be inherited downward from the parent type, while other attributes are defined specifically for a particular subtype.

d. Semantic Database Model

The *semantic database model (SDM)* [Ref. 100] uses the aggregation and instantiation abstraction concepts and distinguishes between entities, which are nonatomic abstract objects, and names, the identifiers for atomic objects. This model supports types, which are disjoint classes of objects, and subtypes, which may overlap. SDM also employs a *grouping type*, which is formed by treating instances of a type as subtypes. Grouping types allow relationships between sets of subtypes having a common parent type to be created and named. Relationships are also permitted to have attributes associated with them. Attributes in SDM can be defined as single- valued or multi-valued.

e. Taxis

The Taxis [Ref. 91] data model was developed to support the information system design process. Taxis uses an object-oriented framework, where each object in the model represents a real-world (application) entity or concept, and employs the aggregation, classification, and generalization abstraction concepts. *Transactions*, which are groups of primitive operations, are used in Taxis to model complex activities in the application environment. These transactions can be organized into subclass hierarchies to form higher level procedures.

A compiler was written for Taxis which takes advantage of traditional data management facilities. This implementation decision was intended to decrease the effort required to produce the compiler [Ref. 101]. The compiler translates Taxis programs into Pascal/R, which interfaces to a relational database management system.

f. SAM*

*SAM** [Ref. 86], which is a refinement and extension of the *semantic association model (SAM)* [Ref. 102], includes support for temporal, positional, and procedural relationships, hierarchies of data structures, recursive definition of objects, modeling of multiple versions of an object, and complex data types. This data model distinguishes between *atomic* and *nonatomic* concepts. An atomic concept is one which cannot be decomposed, and is assumed to have a well-understood meaning which does not have to be defined in terms of other concepts. Nonatomic concepts are defined in terms of other atomic and nonatomic concepts.

When atomic or nonatomic concepts are grouped to describe a higher-level non-atomic concept, an *association* is formed. The types of association supported by SAM* can be distinguished according to their structural properties, operating characteristics, and any constraints that users may place on them. Among the associations supported are the membership, aggregation, and generalization associations, which are analogous to the abstraction concepts of classification, aggregation, and generalization, respectively.

g. Extended Relational Model

The *extended relational model RM/T* [Ref. 103] extends the traditional relational model by supporting null values, the aggregation and generalization abstraction concepts, and a richer variety of objects. Types are represented by relations with an internal identifier for each instance of a type. Attributes are also represented by relations with property values for the internal identifiers.

h. Object-Oriented Approach

One of the major distinguishing features of an *object-oriented* system from traditional systems is its ability to handle objects of arbitrary type. Traditional data management systems are limited to objects of type record (they are record-oriented). Object-oriented systems define types to be similar to abstract data types; i.e., the properties and operations for a given type are encapsulated. The classification abstraction concept forms the basis for object-oriented systems, that is, objects are placed into classes based on their properties, and classes are organized into hierarchies which support inheritance.

Most of the object-oriented systems are based on the *Smalltalk* [Ref. 104] programming language. Smalltalk models both entities and relationships as objects. In addition, classes and properties are treated as objects. When a class is defined, the variables (properties) and messages (operations) for that class are specified. Once we know what class a particular object belongs to, we can access the information about the properties and valid operations for that class. Operations are performed by passing messages to objects, which results in a response dictated by that object's properties.

D. SUMMARY

In this chapter we briefly reviewed the traditional data models and discussed their limitations. Their most significant limitation is the lack of semantic expressiveness which is necessary to capture the semantics of advanced application areas such as

manufacturing. We discussed semantic data models and the abstraction concepts which differentiate them from the traditional models.

V. DATA-ORIENTED MODEL FOR INTEGRATING MANUFACTURING FUNCTIONS

A. MOTIVATION

As we stated previously, our approach to integrating manufacturing processes from a data-oriented perspective considers CIM as the composition of a design phase, a production planning phase, a production monitoring phase, and considers the traditional business functions as peripheral to these three phases. Manufacturing processes are associated with one of these phases based on their type of data usage. The way in which the basic processes are grouped into these phases does not affect our proposed integration strategies.

The major advantage of our data-oriented approach over process-oriented approaches is that the integration of product design and manufacturing functions is considered in the context of the manufacturing system as a whole. The database support for the manufacturing environment includes the production of appropriate data as a byproduct of primitive functions such as product design. As soon as a product is designed, the alternative process plans for that product are immediately known. Process-oriented approaches regard integration as the automation of interfaces between existing functions, view these functions in a local context, and do not allow for the possibility that a more natural integration might occur if the product life cycle was redefined.

We will introduce our data model by describing the abstraction concepts it supports. We will first define our model informally using illustrative examples and then define it formally in section C. It will be clear from our description that no other existing data model provides *natural* abstraction support to the CIM environment.

B. DATA MODEL DESCRIPTION

Our model includes the molecular aggregation, generalization, version hierarchy, instantiation, and instance hierarchy abstraction concepts. We believe these are necessary to support the manufacturing process, and therefore are useful for other advanced application areas as well. As we describe our modeling abstractions, we will discuss existing concepts from which they were derived, where appropriate.

Some of the modeling abstractions supported in our data model are portrayed by a *conceptual schema* which the user will manipulate (see Figure 23). The conceptual schema will show the allowable type/subtype aggregations, component relationships, and the acceptable combinations of primitives which can produce higher-level objects. It is in this conceptual schema that the primitives for an application environment are defined.

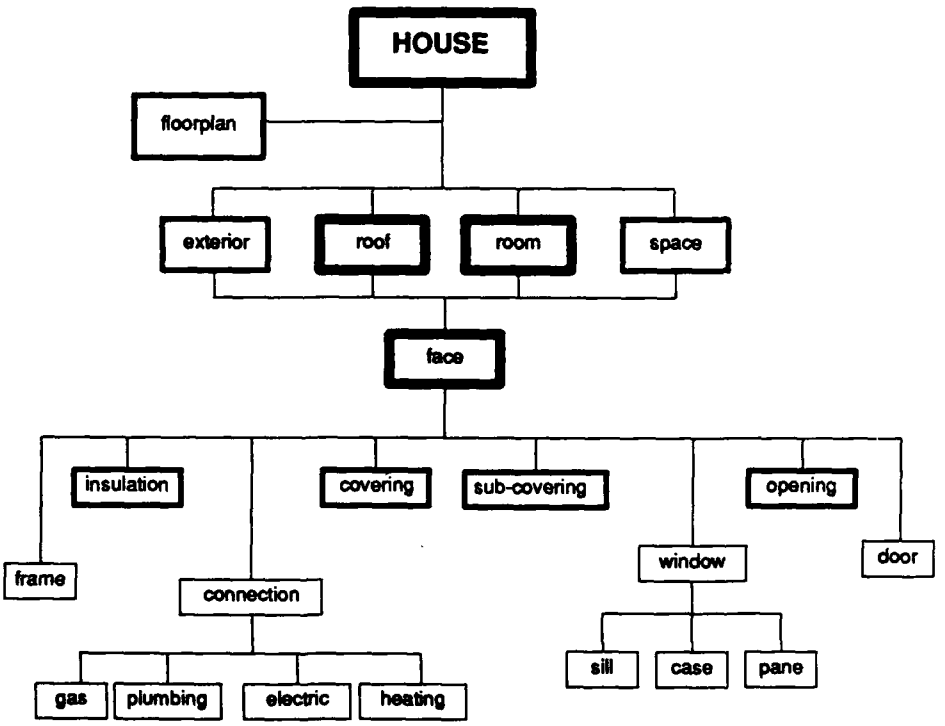


Figure 23. Example Conceptual Schema

Primitives can be defined to any level of abstraction, and can be composite objects themselves. These primitives are the building blocks which the data model manipulates in support of a specific product design, process plan, or other application. A separate schema is produced for each different application to be modeled and manipulated.

Each type and subtype in the conceptual schema will have a *prototype* associated with it. The prototypes will contain *slots* for attribute values, allow default values to be specified, and provide inheritance information. When instances are created, extensions of these prototypes are created, allowing for attribute values to be defined which are unique for that instance.

Figure 23 provides an example of a conceptual schema. This schema represents the hierarchy of type aggregations for a generic **house**. An instance of this schema would contain data for a specific house being designed.

A house could be the aggregation of a **floor plan**, an **exterior**, a **roof**, and interior **rooms** and **spaces**. Each of **exterior**, **roof**, **room**, and **space** are further defined as aggregations of objects, some of which are shared. For example, both **roof** and **exterior** can have a component called **opening**.

The bold rectangle notation represents types which have named subtypes. For example, **room** has subtypes named kitchen, den, bathroom, bedroom, etc., which can be instantiated to produce a specific configuration.

In summary, the conceptual schema provides the medium through which the data model captures the data for a particular application, e.g., product design. Together, the data model and conceptual schema determine the full range of alternatives available in an application.

1. Molecular Aggregation

We will use the aggregation abstraction concept described earlier to support several aspects of the manufacturing environment. For example, in product design, aggregation will be used to model assemblies which are composed of subassemblies and component parts. In production planning, aggregation will be used to form process plans

from individual machine operations and other process plans. In production monitoring, shop floor layouts will be determined by aggregating machine cells of various types.

In general, aggregation will be used to combine intensions and extensions of objects of possibly different types into a higher level object, which will also be an intension or extension, respectively, of a type. Figure 24 shows some sample aggregations using our model.

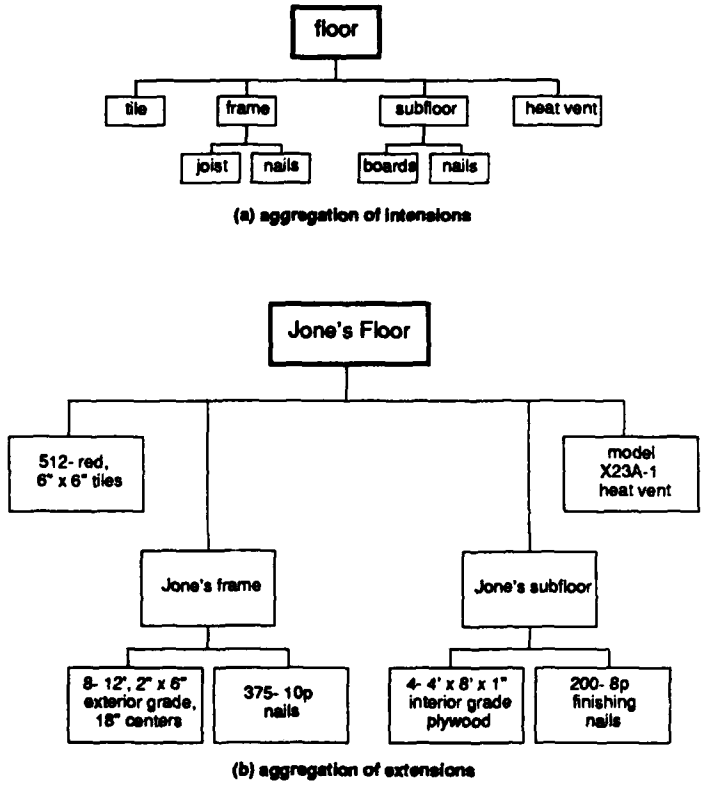


Figure 24. Sample Aggregations

2. Generalization

The generalization concept will be used in our model to provide the relationship between types and their subtypes. Types will be defined as generalizations of

a set of named subtypes, and will be treated as primitives from which versions and instances can be made directly. An example of generalization would be the creation of a type **wood-working machine** from the subtypes **drill press, jointer, table saw**, and **lathe**. The notion of subtype is important to our model because different subtypes (of the same type) will be permitted to have different sets of attributes.

One important aspect of the use of generalization in our data model concerns the inheritance of attributes between related types/subtypes. In Figure 25, **wood-working machine** has been created with attributes **owner** and **power source**. Each of the subtypes **drill press, jointer, table saw**, and **lathe** also have these same attributes, plus other attributes which can be defined uniquely for each subtype. When the subtypes **drill press, jointer, table saw**, and **lathe** are created, their subtype-unique attributes are defined and then the attributes from their generalized type are inherited (in this case **owner** and **power source** are inherited from type **wood-working machine**).

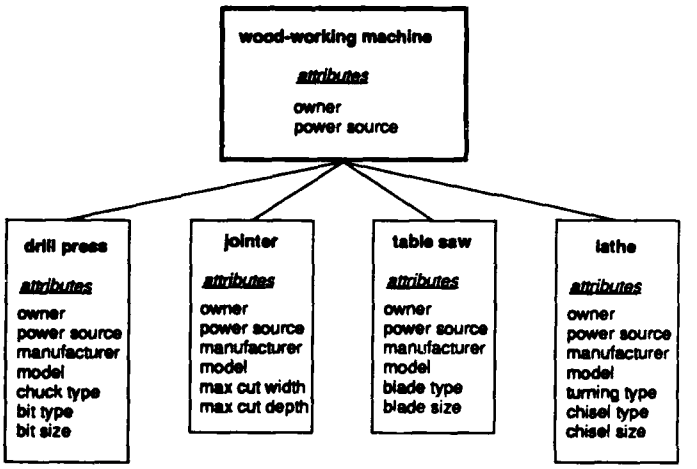


Figure 25. Generalization

3. Version Hierarchy

A version of a type (or subtype) will be defined to be a molecular object with two components, an interface and an implementation. The interface for a version is

specified by listing the properties or attributes which describe it. The implementation for a version is specified by providing values for the interface attributes. In our model, a version will have its interface details completely specified, but its implementation details will be in some stage of completion. This definition allows a version to be plugged, partially plugged, or unplugged [Ref. 92]. Figure 26 shows an object of type **CAR** with an object version 1988 X-Car of type **CAR**. The object of type **CAR** has its interface defined, which is denoted by the topmost block in the figure with the atributes **year**, **make**, **model**, **color**, and **owner** listed. The implementation details for this object are not specified, denoted by the unspecified values for those attributes. Object 1988 X-Car has the same interface details as its object type, and also has some implementation details specified, denoted by the value "1988" for the **year** attribute and the value "X-Car" for the make attribute. In this example, the interface (function) of the object is specified, but the implementation details (e.g., what color is the car?) are not completely specified.

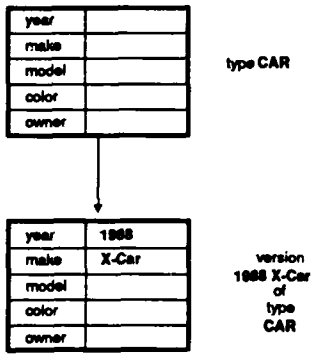


Figure 26. Version of a Type

Versions can have two distinct forms of attributes; those inherited from the object type, and those with unique values for each version. Attributes inherited from the object type reproduce the interface characteristics of the object type. Attributes defined to be version specific are the attributes whose values distinguish one version of a particular type from another version of the same type.

The difference between a version and an instance of a type/subtype is that a version is created at an intermediate point in the modeling of an application, permitting future work to begin at that point, with implementation details partially specified. A type/subtype is considered a starting point in the modeling of an application, with no implementation details specified.

In specifying the various possible values that attributes can take on, the version hierarchy is formed. The purpose of this hierarchy is to expand the set of possible starting points for future work. This notion of a hierarchy of intermediate modeling points is one distinction between our model and those previously discussed. This concept is extremely valuable because it minimizes the amount of redundant work in all aspects of the manufacturing process. Traditionally, such redundancy occurs in product design, where products are repeatedly designed using the same primitive elements; in process planning, where machine operations are constantly refined, creating new process plan alternatives; in shop floor layout, where improvements in efficiency are sought by shuffling resources; and in scheduling, where priorities and resource availability are constantly changing.

Our ability to model versions in this hierarchical manner comes directly from our definition of version. [Ref. 92] defines versions to be objects that have the same interface, but different implementations. Our definition is more general in that the implementation can be specified to any level of detail desired; plugged, or fully specified; partially plugged, or partially specified; or unplugged, in which case no implementation details are specified. The flexibility we gain in generalizing the definition allows us to better model, and more efficiently support, the manufacturing environment.

Our version hierarchy is also different from that described in [Ref. 92], where a hierarchy forms from the aggregation of versions to create higher level versions. Figure 27(a) shows an example of this concept. Our version hierarchy, on the other hand, forms from the specialization of versions to form lower level versions. Figure 27(b) depicts our version hierarchy concept.

Our hierarchy consists of versions which are all of the same type. The versions are related to each other in the manner represented by the hierarchy. All of the versions

are related to their type by version generalization; the topmost version in the hierarchy is directly related and the others are indirectly related. Again, the flexibility provided by our model in representing and relating versions in this way increases the semantic modeling power of the model and brings it closer to the application environment. We know of no other model which supports this construct.

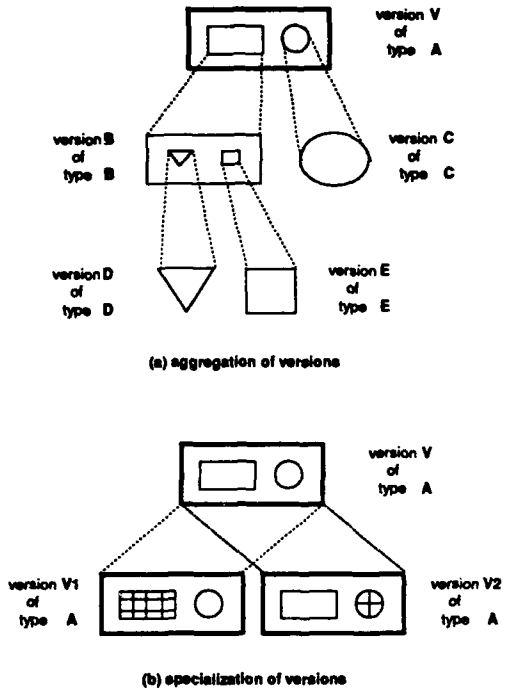


Figure 27. Comparison of Version Hierarchies

4. Instantiation

We use the instantiation abstraction concept in our model for several purposes. The dotted and dashed lines in Figure 28 represent instantiations which create versions and instances of objects, respectively. Both types and versions can be instantiated, and the result can be either a new version or an instance of an object.

Instantiation includes an inheritance mechanism which is more direct than the inheritance associated with the generalization abstraction concept. The instantiation inheritance copies all of the attributes and attribute values of the instantiated object. No new attributes can be defined for the instances created, but attribute values may be further specified. Thus, in Figure 28, the attributes of **version V1**, **instance I1**, etc., are the same as the attributes for **type A** itself. The only difference between any of these instantiated objects, either versions or instances, are differences in attribute values. Two instances of the same type or version, such as **I1** and **I2**, will always be distinguished by the values of attributes, in particular, attribute values which are not inherited during the instantiation process. Therefore, the major distinction between an instance and a version is the same as the distinction between extensions and intensions. In the manufacturing environment, instances are meant to represent real-world products, process plans, schedules, etc., while versions serve as templates which define those real-world objects to some level of detail.

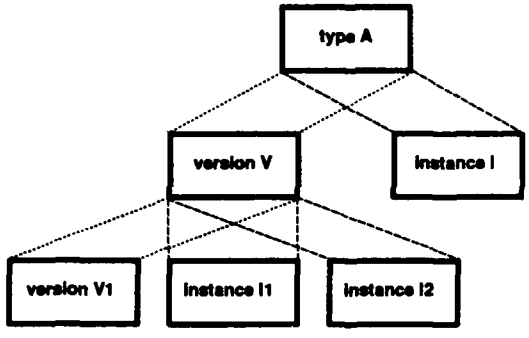


Figure 28. Types of Instantions

5. Instance Hierarchy

We created the instance hierarchy to supplement the other abstraction concepts in our data model, providing a mechanism which allows a user to maintain all of the different instance alternatives for a particular function. For example, a design engineer could keep all of the variations for a product being designed. Similarly, alternative process plans could be kept this way until a final plan was decided on.

Figure 29 depicts the sequence of events that might occur as a design engineer interacts with our data model.

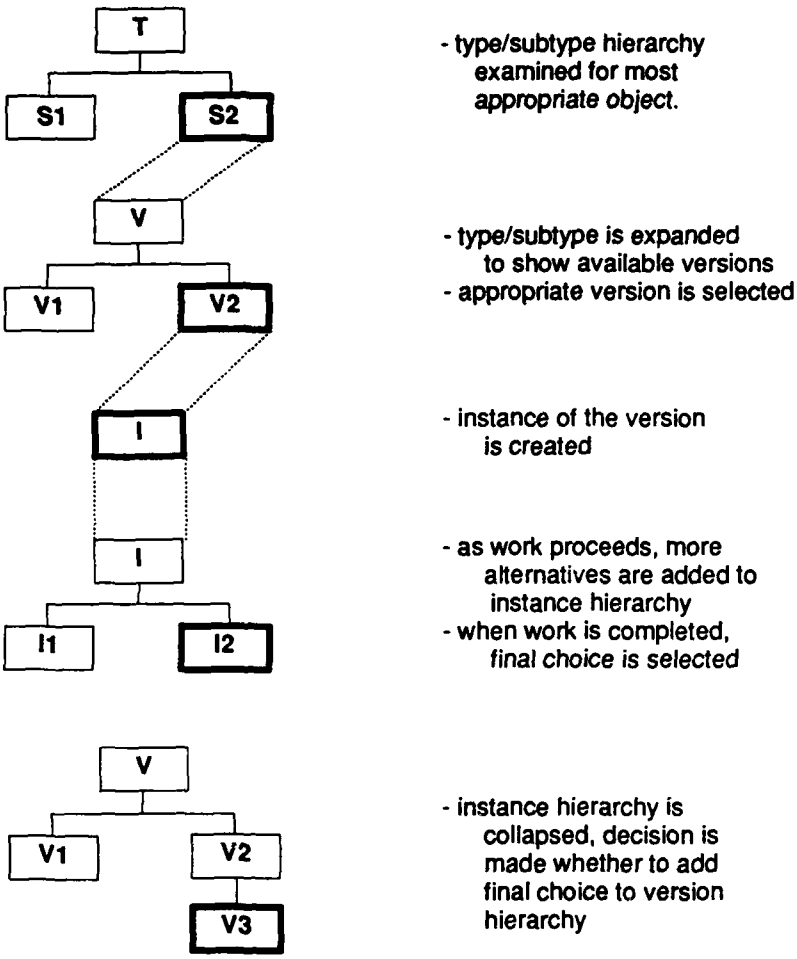


Figure 29. Operation of Data Model

Note that the instance hierarchy, like the instance itself, is a temporary entity within the system. When a designer decides which alternative in the hierarchy will become the final choice for a given product, the hierarchy collapses, leaving only the selected alternative. The design is archived with the version from which it was created. The designer then has the option of creating a new version from the new design, which

will be added to the version hierarchy (in the appropriate place) to become a starting point for future work. If the new design is added to the version hierarchy, another decision has to be made regarding which attribute values will be included when the version is created.

C. FORMAL DEFINITION OF THE DATA MODEL

We will use standard mathematical notation to define our data model. We will start by defining the notions of type and object. We will then define each of the abstraction concepts used in our model in terms of these notions.

We define a *type* to be the characterization of a set of values and the set of operations that are applicable to those values. We further define a type to include *system-defined* and *user-defined* types. System-defined types are the primitive types integer, real, character, string, etc., found in most database systems. User-defined types are formed by aggregation of previously defined types, each of which may be either system-defined or user-defined. We will denote a type in this discussion by the use of capitalized letters.

The aggregation operation, used to create a type P from types T_1, T_2, ..., T_n is defined as follows:

$$P = \mathbf{Agg}(T_1, T_2, ..., Tn) \Leftrightarrow P = \overset{n}{\underset{i=1}{\times}} T_i$$

where $\times$ denotes the Cartesian product operation, $T1 \times T2 \times ... \times T_n$. Therefore, a user-defined type is the Cartesian product of the sets of values which are the aggregates for that type. The aggregates for a type P, denoted by **Agg**(P), are defined as follows:

$$\mathbf{Agg}(P) = \{T_i \mid P = \overset{n}{\underset{i=1}{\times}} T_i \}$$

We define an *object* to be a member of a type, or in other words, a value in the domain of a type. We will denote objects by use of bold-faced lower case letters. Using

our notation, an object $\mathbf{o} \in T \Leftrightarrow \mathbf{o}$ is of type T. The aggregates for an object **o** are defined as follows:

$$\mathbf{Agg(o)} = \{\, \mathbf{a}_i \mid \mathbf{o} \in T \wedge \mathbf{a}_i \in Ti \;\; \wedge T = \overset{n}{\underset{i=1}{\times}} T_i \,\},$$

that is, the aggregates of an object of type T are the objects of type T_i , where T_i is an aggregate of type T.

The generalization of subtypes S_1, S_2, ..., S_n, denoted by **Gen**(S_1, S_2, ..., S_n), to specify type T is defined as:

Gen(S_1, S_2, ..., S_n) = T $\Leftrightarrow$

$$(A \in \mathbf{Agg}(T) \Rightarrow (A \in \mathbf{Agg}(S_1) \wedge A \in \mathbf{Agg}(S_2) \wedge \ldots \wedge \mathbf{A} \in \mathbf{Agg}(S_n))).$$

The specialization relationship between a type T and its subtypes is defined as:

$$\mathbf{Spec}(T) = \{S\} \Leftrightarrow (A \in \mathbf{Agg}(T) \Rightarrow A \in \mathbf{Agg}(S))$$

and a subtype S of type T is defined as

$$S \;\mathbf{St}\; T \Rightarrow S \in \mathbf{Spec}(T).$$

An instance of a type is defined as follows:

$$\mathbf{x}\;\mathbf{In}\;Y \Rightarrow (A \in \mathbf{Agg}(Y) \Rightarrow (\exists\, \mathbf{a} \in \mathbf{Agg}(\mathbf{x}) \ni (\mathbf{a} \in A \vee \mathbf{a} = \phi))).$$

In our notation, **x In** Y reads "**x** is an instance of Y" and ϕ denotes a null, or unspecified value..

A version v of type T, denoted by **v Ver** T, is defined as:

v Ver T $\Rightarrow$

$$(A \in \mathbf{Agg}(T) \Rightarrow (\exists\, \mathbf{a} \in \mathbf{Agg}(\mathbf{v}) \ni (a \in A \vee \mathbf{a} = \phi)))$$

$$\wedge\, (\exists \mathbf{y} \in \mathbf{Agg}(\mathbf{v}) \ni \mathbf{y} = \phi).$$

An instance of a version is defined as follows:

$$\mathbf{x}\;\mathbf{In}\;\mathbf{y} \Rightarrow (\mathbf{a} \in \mathbf{Agg}(\mathbf{y}) \Rightarrow \mathbf{a} \in \mathbf{Agg}(\mathbf{x})).$$

Our version hierarchy requires two definitions. The first relates a version to the type from which it was created. The second relates a version in the hierarchy to the version from which it was created.

$$(2)\ v_1\ \mathbf{Vh}\ v_2 \Rightarrow ((y \in \mathbf{Agg}(v_2) \Rightarrow y \in \mathbf{Agg}(v_1)) \wedge (\mathbf{Agg}(v_1) \neq \mathbf{Agg}(v_2)))\ .$$

Our definition of instance hierarchy is as follows:

$$i_1\ \mathbf{Ih}\ i_2 \Rightarrow (y \in \mathbf{Agg}(i_2) \Rightarrow y \in \mathbf{Agg}(i_1)) \wedge (\mathbf{Agg}(i_1) \neq \mathbf{Agg}(i_2))\ .$$

D. ROLE OF THE DATA MODEL

The purpose of a data model is to define the rules according to which data are structured [Ref. 79]. The major way of structuring data is through the use of abstraction. Using abstraction, the general properties of objects are emphasized while their details are suppressed. The use of data modeling techniques in advanced application areas such as Computer Integrated Manufacturing serves an additional purpose. The data model, if properly developed, takes on an important role in the attempt to automate and integrate otherwise autonomous functions. The data model itself serves as a standard which facilitates integration.

In the design process, the data model could provide a standard which different product designs can use to ensure compatibility in the later stages of production. In particular, this standard will facilitate the integration of design data into the process planning and scheduling functions. The role of the data model in process planning will be to provide a standard which different product process plans can use to ensure compatibility with and facilitate integration into the scheduling function. If the same data model is used to support design, process planning, and scheduling, then the compatibility between design and process planning functions could be extended to the scheduling function, providing a natural form of integration of the major components of the manufacturing process.

E. SUMMARY

In this chapter we presented the data-oriented manufacturing model. The model was described as the composition of several data abstraction concepts presented in Chapter IV.

We provided a formal description of the data-oriented manufacturing model and described the role that the data model plays in enforcing standards in a system.

VI. HIGH-LEVEL INTERFACE APPROACH TO INTEGRATING MANUFACTURING FUNCTIONS

A. MOTIVATION

The goal of integrating manufacturing functions is not easy to achieve. A major problem to be overcome is the decision on which strategy is to be used. The investment in existing resources cannot be overlooked in planning an implementation strategy. As we stated previously, one way to consider the existing resources in moving toward the implementation of Computer Integrated Manufacturing is to use the high-level interface approach described in Chapter III. Since this approach still only solves the problem locally, and does not view manufacturing functions in the context of the entire system, it is considered a short-term solution. Since the cost of implementing a fully integrated system will be too much for many manufacturers to bear, this high-level integration may be their best approach.

We will demonstrate this integration concept by describing a high-level interface between Computer Aided Design and Computer Aided Manufacturing, using the partition of functions from Figure 3. Using our data-oriented approach to integration, we will focus on the data requirements for integrating CAD and CAM.

B. DATA REQUIREMENTS FOR INTEGRATING CAD AND CAM

Figure 30 depicts the data interactions in CAD and CAM. CAD uses the conceptual schema (as discussed in Chapter V) and the data model to produce the appropriate design data for the product to be manufactured. The design data is used in the industrial engineering function to produce bill-of-material and manufacturing operations information, which is used in CAM to determine how and when the product will be produced. Since our main objective is to provide an interface between CAD and CAM, we will propose to use the design data produced by CAD to automatically produce the

bill-of-material and machine operation information required in CAM to develop a production schedule.

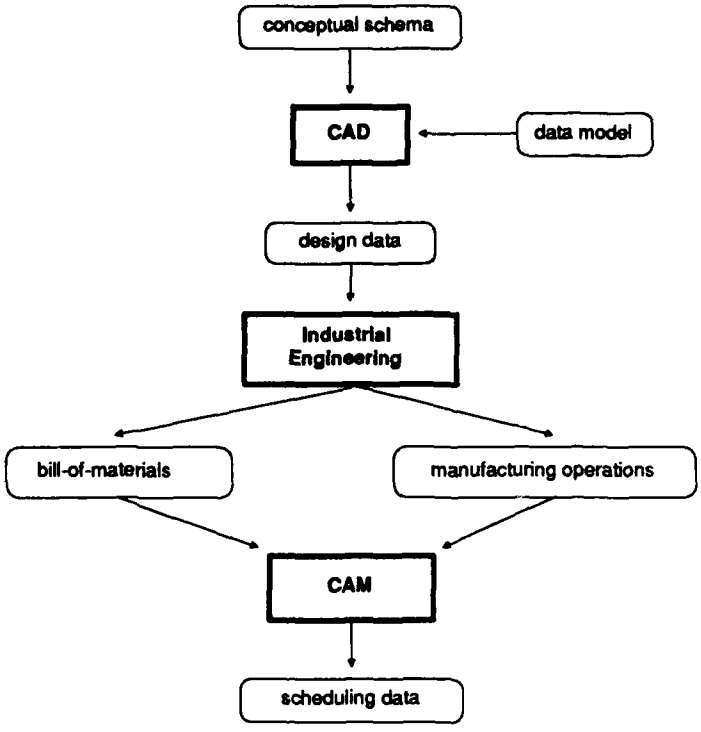


Figure 30. CAD/CAM Data Interaction

1. Representing Design Data

The CAD process, guided by the data model, records actual instantiations of the primitive types represented in the conceptual schema to form the *design schema* for the product being designed. The design schema for a product uses the inheritance mechanisms from the data model to infer some attribute value information about the properties of primitive types/subtypes from known information about related types/subtypes. Our design schema uses both **part-of** and **contains**, which are aggregation relationships, to pass information up and down the hierarchical structure.

a. Use of Prototypes and Inheritance

Each type/subtype in the conceptual schema has a corresponding intensional prototype associated with it. A prototype is a block of memory allocated to store data using the aggregation abstraction concept. As the conceptual schema is manipulated to create the design schema, intensional prototypes are instantiated to capture the design data associated with the use of an object of a given type/subtype in the design. Figure 31 is an example of a prototype (intensional) for type **cover**. Each instantiated prototype has named slots which can be filled with either relationship or property data. The slots **part of** and **contains** are used in the prototypes to represent relationship data. In Figure 31, **part of** relates the **cover** to a particular **face** (using the conceptual schema shown in Figure 23). **Contains** is used to identify prototypes at the next lower level in the conceptual schema, and stores data in the form of a list, so that a variable number of relationships can be represented. The slot **material type** is used to hold property data, in particular, the kind of material that the **cover** is made of. Note, as shown in Figure 31, that some slots are marked with an *, signifying that inheritance can be used to provide a value, while other slots may be marked with **, denoting an attribute whose value is optional. The slots which are optional are those that could have a nonsensical value under some circumstances. For example, the **depth** of a **cover** of paint would not normally be specified, while the **depth** of a **cover** of panelling would be.

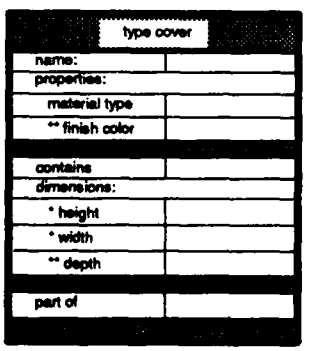


Figure 31. Sample Prototype

Careful thought must be given to the use of optional slots when designing prototypes. The design and efficiency of the CAD/CAM interface may be affected by the improper use of optional slots since the interface has to determine for each use of the prototype whether or not the optional slots should be filled. The greater the number of optional slots, the greater the complexity of the interface becomes. Another aspect of prototype design which must be considered concerns the format of the slots. The format should be kept as simple as possible to minimize the effect on the interface design. Value information for each slot should either consist of two parts, a measurement value and the units of measurement, or a single part, the property value. The description of the slot, e.g., **height** in Figure 31, may be specified in any way, but should have standardized usage throughout the prototypes for a particular application.

b. Coordinate Systems

In order to specify location data in a prototype, it is imperative that the frame of reference be known by any process using that data. For most circumstances, three frames of reference should suffice, *global* or world, *product*, and *local* coordinate systems. Figure 32 depicts the relationships among these coordinate systems.

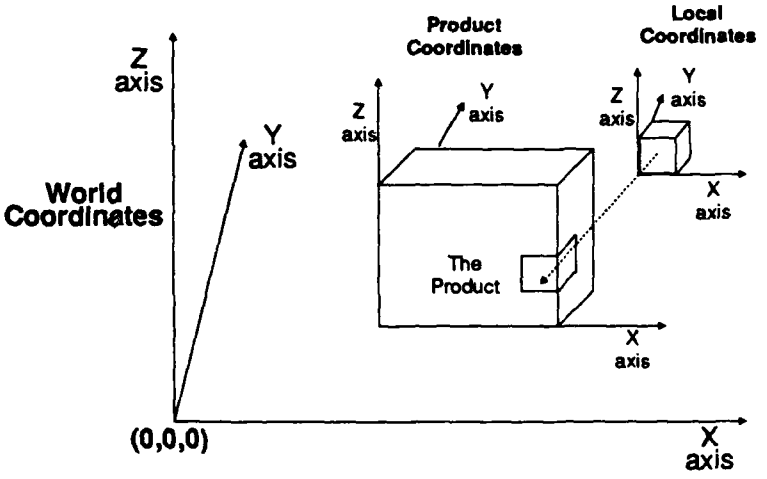


Figure 32. Coordinate Systems

Global coordinates relate an object location in the real world (on planet Earth). The X and Y axes could represent the latitude and longitude, respectively, and the Z axis could be perpendicular to the ground, to represent the elevation of an object with respect to sea level. The product coordinate system expresses information relative to the object itself, and is useful when locating components of the object, regardless of the location of the object in global coordinates. The local coordinate system extends the product coordinate system so that subassemblies of an object in product coordinates can have their own coordinate system to relate components of the subassembly to the subassembly itself.

The uses of product and local coordinate systems not only eliminates the need for global coordinate information under most circumstances, but also provides automatic update of location information during design changes. For example, if a wall, containing a window, is moved, and the window location is specified in a local coordinate system relative to the wall, then the window coordinates need not be modified to reflect the change in position of the window with respect to the overall product, or the real world.

One other valuable piece of information is used to specify the *orientation* of an object. The *normal* is defined to be a unit vector perpendicular to the surface of an object. If many flat parts are being used in a product, then the normal can be used to gather additional information about how the parts are related in the overall product. For example, a wall can be distinguished from a ceiling or floor, which can also be distinguished from each other, using normals. Figure 33 shows an example of a normal.

Figure 33. Example of a Normal

By definition, each flat surface will have a unique normal with three components, one for each of the three dimensions in the coordinate system. A component value of 1 indicates that the normal is parallel to the axis in question, with an orientation in the positive direction of the axis. A value of -1 indicates that the orientation is in the negative direction of the axis, still parallel to the axis. The value of each component is equal to the cosine of the angle between the normal and its axis [Ref. 105], and therefore, will always be between -1 and 1.

c. Storage and Manipulation of Design Data

The use of prototypes to capture design data has a major advantage in addition to those already mentioned which affect the CAD/CAM translation process. The representation of design data in prototypes, as a tabular array of data, permits storage and manipulation (insertion, modification, deletion, etc.) of the attribute values using a *non-first normal form* relational model [Ref. 106] at the physical level. The non-normal form relational model is required because the values of the **contains** attributes are contained in a list, which is not *atomic*, and therefore violates the requirements of the normal relational model.

Using the non-normal form relational model, a database scheme is developed containing one relation for each type/subtype in the conceptual schema. As prototypes are instantiated during the design process, the slot value information provided by the user is stored in the appropriate relation as a tuple. Since each instantiated prototype contains a unique name, specified by the **name** slot, that name can serve as the identifying key (primary key) for its associated relation.

2. Data Used in CAM

The industrial engineering activity, shown in Figure 30, converts the design drawings and other design information into working papers for manufacture. These working papers define *what has to be produced* and *how it should be manufactured.* When the industrial engineering activity is completed, the sequence of manufacturing

operations necessary to produce the product and the raw material requirements for those operations, in the form of a bill of materials, will be known.

a. Manufacturing Operations

To plan the sequence of manufacturing operations necessary for production of a product, the product is decomposed into a series of operations which are related to each other both temporally and spatially. These temporal and spatial relationships determine the sequence of the operations. In the decomposition process, the most suitable machining technologies are considered at each step. Decisions are made regarding the forming, shaping, cutting, etc., of raw materials, and the appropriate machinery, tools, and fixtures are selected which perform the desired operation. The information on the sequence of manufacturing operations will drive the CAM functions, in particular, the scheduling of resources to perform those operations for a product within a desirable time frame.

b. Bill of materials

The raw material requirements planning aspect of industrial engineering determines the quantity of materials required for a product, using information about the external and internal composition of that product. The cost of each component can be used to calculate the materials cost for the product. The bill of materials produced will show these costs and the relationships of components within subassemblies and subassemblies within the overall product.

C. EXPERT SYSTEM TRANSLATOR

Our approach to a high-level interface of CAD and CAM is to develop an *expert system translator*. The basic task of this translator is to automatically conclude the quantities, types, and assembly sequences of raw materials needed to manufacture a product from design data. In addition, the translator will provide for resolution in the event that it receives conflicting data. An example would be the preference of standards data over design data, in situations where standards would otherwise not be met. The

translator will be opportunistic, that is, it will use substitution criteria whenever possible to lower cost without sacrificing quality. The translator will use the schema and assembly data as input to the backward-chaining control mechanism. In addition, the translator performs various standards checks on the CAD data to ensure its correctness. Correctness is used here to imply that the data meets all known requirements. These requirements may be based on laws of physics, laws of government, or anything else deemed appropriate.

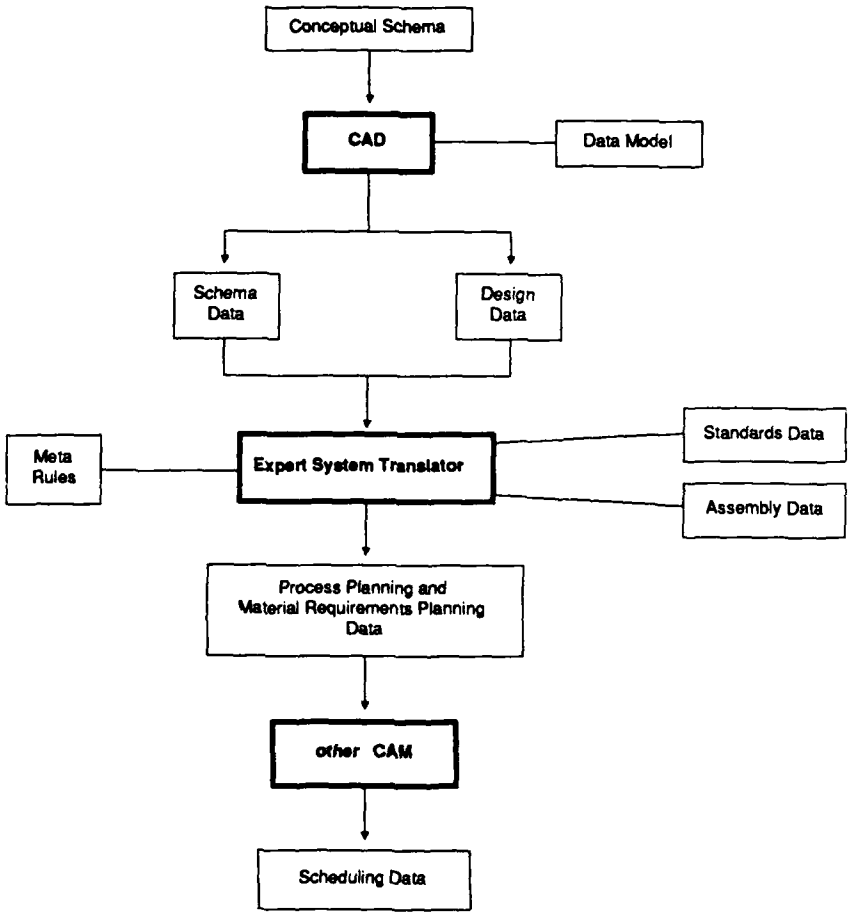


Figure 34. CAD/CAM Translation

Figure 34 depicts the data involved in the CAD/CAM translation process. Since our main objective is to automate the industrial engineering process which produces process planning and material requirements planning data, we will not concern ourselves with data requirements beyond the scheduling process. The design phase is represented by the box labelled **CAD** which takes the **conceptual schema** as input and outputs **schema** and **design** data using the **data model** as a guiding mechanism. The **expert system translator** uses the **schema** and **design** data as input and produces **process planning** data which is used in the manufacturing phase, eventually being converted to **scheduling data**. We will briefly describe expert systems and then discuss each of the data pools shown in Figure 34 individually and tie them together by describing the interactions which occur.

1. Expert Systems

The term *expert system* has been used in our discussion of the CAD to CAM translator. True expert systems are written using artificial intelligence languages such as Prolog and belong to the class of artificial intelligence applications known as knowledge-based systems [Ref. 107]. Figure 35 shows the structure of an expert system.

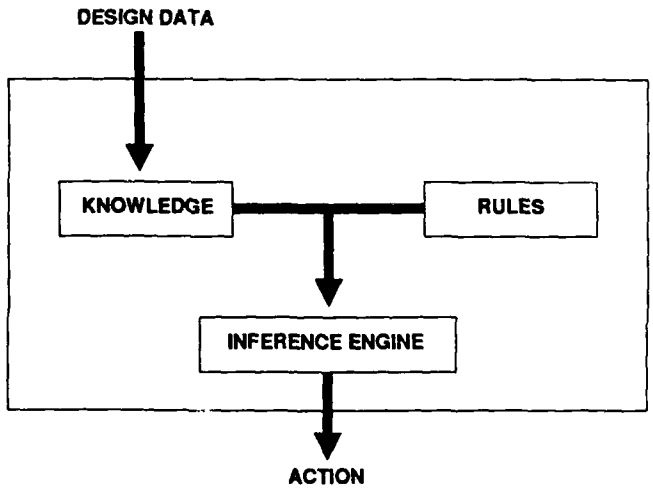


Figure 35. Expert System Architecture

Using Prolog programs, each rule represents a portion of an expert's knowledge of the problem domain. This knowledge is reduced symbolically to facts about the surrounding environment. Data supplied to the expert system is then treated as facts and used to infer new facts. One interesting feature of some of these rules is their apparent link to rules of thumb, known as *heuristics* [Ref. 107].

Among the necessary qualities of an expert system is the requirement that it properly perform its assigned tasks. A potential problem is that experts may not agree on what is proper. For example, consider two builders each constructing a house of similar design. While most of the assembly priorities would be similar, the experts could disagree on matters such as the density of fasteners required for a particular kind of material. One advantage gained by using artificial intelligence techniques is the flexibility to allow experts to set their own priorities by modifying the expert system rule base without any changes to the other parts of the system concerned.

Expert systems are also advantageous in that they have the ability to explain their path of reasoning, although in today's systems the explanation is usually nothing more than a trace of the rules proven to be applicable.

One consideration in the development of an expert system is that the code be partitioned according to the functional areas of concern. This is important because many experts are limited in the breadth of their knowledge. Therefore, the code should be divided in such a way that each expert has responsibility for that portion of the system pertaining to his area of expertise.

2. Translator Implementation

A simple one room house is used to demonstrate our expert system translator. Figure 36 shows a design schema for this example.

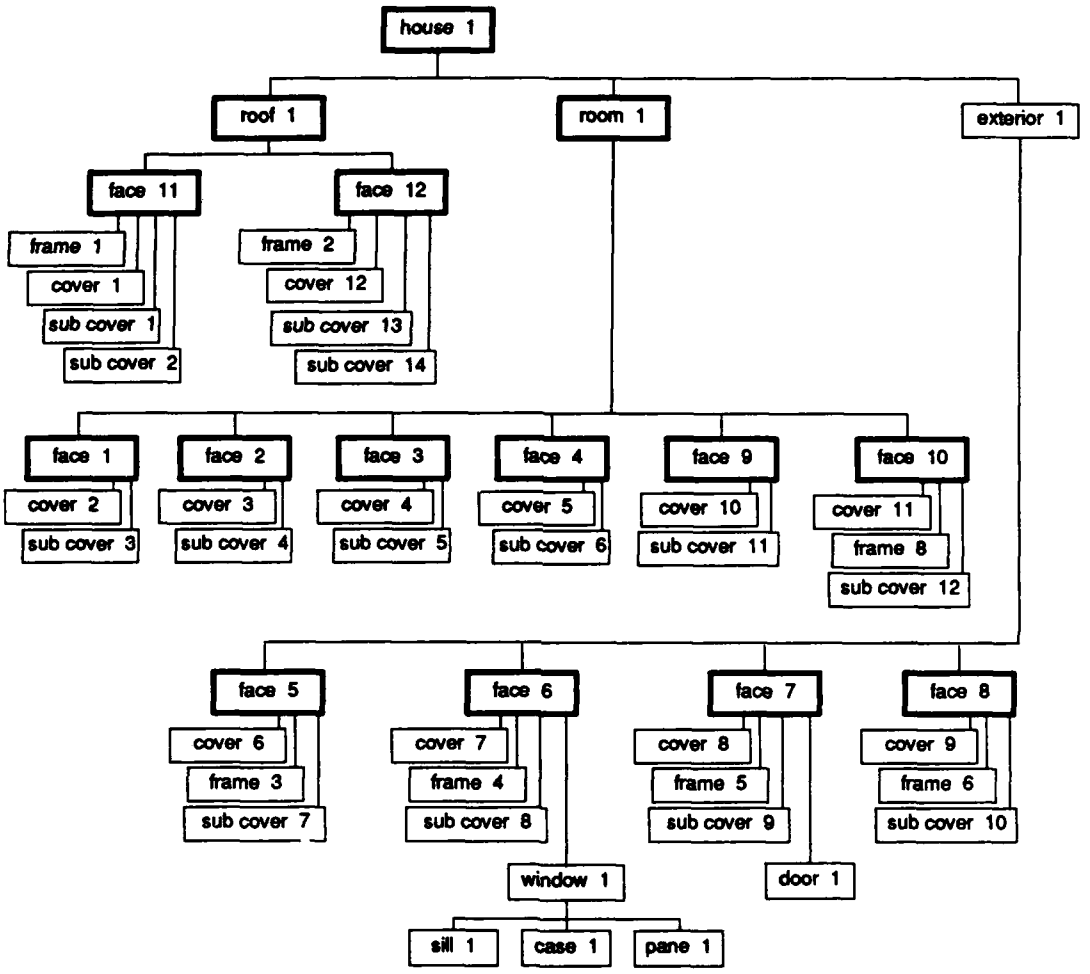


Figure 36. Design Schema

a. Schema Data

The schema data consists of semantic network-type relationship information from the conceptual schema for a particular product. This schema data will be used by the expert system translator to associate design data according to the conceptual schema relationships. The relationships supported by our system are the KIND-OF (instance) and PART-OF (aggregation) [Ref. 108]. The PART-OF relationship provides an attribute inheritance mechanism whereby the system can infer attribute values in cases where those values were incompletely specified by the designer. Inheritance begins at the closest ancestor and continues up the ancestral hierarchy until a value is found. The relationships in the conceptual schema are stated in the form of facts, as shown in Figure 37. Our system distinguishes between schema data and the conceptual schema because the separation of these allows a user to modify the original conceptual schema in the design process by modifying the schema data without having to change the schema itself. This adds flexibility to the system and permits the conceptual schema to be implemented independently (i.e., it can be represented in a form most appropriate for processing by CAD) of the schema data which will be used by the expert system translator. If no modification is made to the conceptual schema during the design process, the schema data does not have to be re-generated for each product.

```
part_of (house, floorplan).     part_of (house, interior).
part_of (house, shell).         part_of (house, roof).
part_of (interior, story).      part_of (story, room).
part_of (story, space).         part_of (room, face).
part_of (space, face).          part_of (face, sub-cover).
```

Figure 37. Conceptual Schema Data

b. Design Data

The design data consists of the instances of the prototypes created during the design process. All slots are filled in, with default, inherited, or specified attribute values. As prototypes are instantiated, KIND-OF facts are asserted which associate the instance with the type from which it was created. Figure 38 shows an instance of a prototype of type **face** and the design data which corresponds to it. The **kind_of (face1, face)** fact tells the translator that this instance, named, **face1**, is of type **face**. Note the close correspondence between the design data, written in Prolog, and the instance of the prototype. The properites or attributes from the prototype become predicate names and the values of the attributes become arguments to those predicates. In the case of

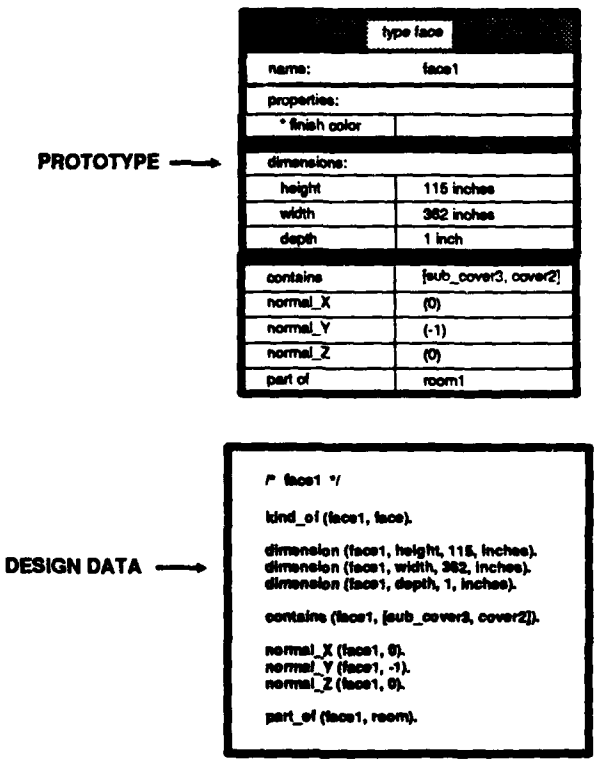


Figure 38. Prototype/Design Data Relationship

dimensions, the names **height**, **width**, and **depth** are treated as attribute values, as shown in the design data.

The prototype shown in Figure 38 can be considered to be complete, even though no value for **finish color** is shown. The value of **finish color** will be determined during the translation process using inheritance. Since **face1** contains **cover2**, the inheritance mechanism will look at the design data for **cover2** and copy the value of the **finish color** given in that portion of the design data. When all of the instantiated prototypes are completed, the design process itself is considered to be complete, and the design data is ready to be used by the expert system translator.

c. Standards Data

Design and manufacturing systems have to take into account a wide variety of Federal, State, local, Occupational Safety and Health (OSHA), quality assurance, and other standards prior to manufacturing a product. For example, a design could call for a 1/4" inside diameter pipe in a specific location, but a local building code may specify a 3/8" minimum inside diameter. In this case, the design specification must be changed to reflect the regulatory requirement. For a given product, thousands of interactions are possible between existing standards and specifications generated from the design process.

These standards are represented in the system by Prolog-style rules to facilitate their enforcement by the expert system translator. Figure 39 gives an example of the implementation of a regulatory requirement.

The **maximum** and **minimum** facts shown on the first two lines provide the limits for a particular type of pipe. The **passed** predicate indicates that the minimum and maximum values with their respective units will be checked against the design values, indicated by the variable **Z** and units variable **Units**. The **convert** predicate converts the standards units of measure to the units in which the design object is measured. The **check_standards** predicate would compare all three measurements to a common unit of measurement and verify that the standard was met.

```
maximum (pipe, plastic12, diameter, 3, inches).
minimum (pipe, plastic12, diameter, 1, inches).
passed (pipe, Type, Dimension, Z, Units) :-
    minimum (pipe, Type, Dimension, X, Unitx),
    maximum (pipe, Type, Dimension, Y, Unity),
    convert (X, Unitx, Min, Units),
    convert (Y, Unity, Max, Units),
    check_standards (pipe, Type, Dimension, Z, Units, Min, Max).
```

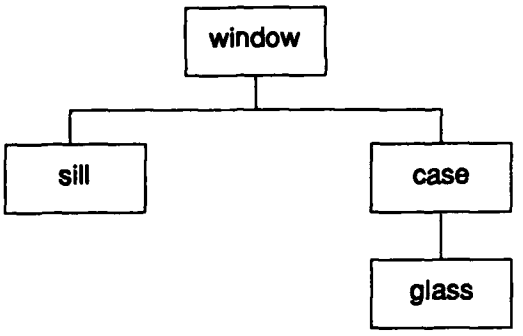
Figure 39. Regulatory Requirement

d. Assembly Data

Assembly data includes sequencing information for assembly of composite objects, or subassemblies, according to the relationships shown in the conceptual schema. This assembly data covers all conceptual schemata for a given application domain. In addition, information on standard material types and acceptable substitutes is included, with their costs. A system could take advantage of fluctuating costs with the substitution information to produce an optimal cost product.

The sequencing information will be represented in Prolog-style rules. Figure 40 provides an example of a portion of a conceptual schema with the sequencing rule to be included in the assembly data for the given product. The first operation fact to be asserted provides for inserting the **glass** into the **case**. The second operation inserts the **case** into the appropriate **sill**. Note that operation information includes details of specific **sills**, **cases**, and **glass**. The assembly rule will produce a set of operation facts for each window defined in the design. Each window will be separately identifiable.

In the object-oriented approach, the assembly rules would be considered part of the operations encapsulated with each data type. We choose to separate these rules for the following reasons. First, the separation allows us to abstract out the




```
assemble (W, window) :- property (W, window, Wtype),
    part_of (W, S), part_of (W, C), part_of (C, G),
    property (S, sill, Stype), property (C, case, Ctype),
    property (G, glass, Gtype),
    assertz (operation (Ctype, assemble, glass, Gtype)),
    assertz (operation (Stype, assemble, case, Ctype)), fail.
```

Figure 40. Component Relationships and Assembly Rule

implementation details so that the conceptual schema isn't tied to the rule-based implementation imposed by the assembly data. The separation also functionally aligns the conceptual schema and assembly data with the people responsible for maintaining them.

The conceptual schema can be developed by users with little technical expertise or familiarity with the implementation considerations necessary to manufacture a product. The assembly data can be maintained by the manufacturing experts who are familiar with implementation details, material properties that may lead to more cost effective substitutions of components, and the sequences of operations used in the manufacturing process. Another reason we separate them is that they serve different functions. The conceptual schema is used by designers, while the assembly rules are part of the expert system translator. The conceptual schema represents one product while the assembly data represents all the conceptual schemata in the application domain. The

assembly data could also contain information about the way the factory chooses to do assembly, which is independent of any particular product.

e. Translator Meta-Rules

The translator meta-rules, combined with the standards data, assembly data, and schema data, will determine how the design data for a particular product will be transformed into process planning data. These rules will enforce the standards given in the standards data, and provide the actual translation mechanism which produces the process planning data. Figure 41 provides a sample of meta rules for the house design and construction example. These meta rules will assert new facts which represent requirements for specific raw materials. Note that the materials list is refined for items such as paint, nails, caulking, etc., whose requirements are expressible as a function of the dimension of the object.

```
raw_materials_needed :-
    kind_of (Extens, Intens),
    property (Extens, finish_type, Material),
    property (Extens, finish_color, Fcolor),
    liquid (Material, Ltype, Covers, Cunits, Lunits, Cost),
    dimension (Extens, height, Ht, Htunits),
    dimension (Extens, width, Wd, Wdunits),
    convert (Ht, Htunits, Height, Cunits),
    convert (Wd, Wdunits, Width, Cunits),
    Area = Height x Width x 2,
    Amt_needed = Area / Covers,
    Tot_cost = Amt_needed * Cost,
    assertz(liquid_list (Material, Ltype, Fcolor, Amt_needed,
            Lunits, Tot_cost), fail.
```

Figure 41. Sample Meta-Rule

f. Process Planning Data

The main outputs of the translator will be a bill of materials and sequencing information about the manufacturing operations required to produce a given design object. The bill of materials will contain information on the assembly of components into subassemblies and quantities of raw materials required for manufacture of component parts. Both of the outputs of the translator fall into the category of process planning and material requirements planning information. The two outputs combined provide all the necessary information for the manufacturing of a product.

g. Scheduling Data

After the requirements for a new product have been determined, the new requirements data can be combined with existing production requirements in the scheduling phase. At this point, priority information is used to determine how to integrate the new requirements into the existing workload. The scheduling data includes assembly data which will be used to coordinate construction of subassemblies with production of components and ordering of raw materials and purchased parts.

h. Operation of the Translator

The following description is based on the execution of the translator using an example included in the translator program listing in Appendix A. Appendix B contains the output produced by the translator for this example. The expert system translator performs a variety of tasks in providing the interface between CAD and CAM. The first task is to use the inheritance mechanism discussed previously to fill in any attribute values not explicitly specified during the design process. At the same time, the design data is validated against applicable standards. If the translator encounters incomplete design data, the attribute name for the missing data is highlighted along with other identifying data so that the appropriate data can be added to the system. The translator will check all of the design data and will only continue on to the next task when no exceptions are detected.

The next task to be performed by the translator is to determine the sequence of manufacturing operations necessary to produce a given product. It is possible during this portion of the translation process for problems to arise which prevent a product from being properly manufactured. The major problems encountered are caused by a different type of incomplete design data. That is, not incomplete prototype information, but cases where additional prototype instantiations should have been included in the design process but were inadvertantly omitted. An example would be a case where a house is designed with all of the required components except for the floor. The translator would detect this type of omission and require a correction before the final step in the translation process is performed.

The last step to be performed by the translator is the determination of the raw material requirements for a product. The translator produces a complete listing of materials, their quantities, and costs. The translator extends this output by considering possible material substitutions and generates a new raw materials list for each substitution it considers.

We will now discuss the operation of the translator in more detail, continuing with the previous one-room example.

(1) Standards Checks

The first series of operations performed on the input data by the translator are those necessary to verify that all applicable standards requirements are met. Figure 42 contains some of the standards that were used for our one room house.

Note that while the **width** and **height** standards for doors apply only to a door of type **door1**, the **depth** standard for doors and the window pane **quality** standard apply to all doors and windows respectively. This demonstrates the flexibility of the language and our system.

In addition to actual physical checks, two other types of standards data are also contained in the standards file. These are shown in Figure 43.

```
minimum (door, door 1, width, 32, inches).
minimum (door, door 1, height, 6, feet).
maximum (door, door 1, width, 4, feet).
maximum (door, door 1, height, 7, feet).
minimum (door, _, depth, 2, inches).
maximum (door, _, depth, 3, inches).

minimum (pane, _, quality, 3).
```

Figure 42. Standards Checks

```
comment (masonry, 'approved methods must be used for building
    masonry walls when outside air temperature drops below 40
    degrees fahrenheit').
comment_for (cover, brick, masonry).
comment_for (cover, concrete_block, masonry).
comment_for (sub_cover, brick, masonry).
comment_for (sub_cover, concrete_block, masonry).

comment (framing, 'grade marks must be clearly visible on all
    framimg members for inspection').
comment_for (frame, wood, framing).

check_for (sub_cover, tar_paper, [tar_paper1, tar_paper2,
    tar_paper3]).
```

Figure 43. Standards Data

The first type of standards data is the **comment_for** data. For example, consider the rule **comment_for(frame,wood,framing).** This rule relates any frame made from a wood product to the comment framing. This allows data that can only be verified during manufacturing to be output by the translator for use in process planning.

The other data type contained in the standards file is the **check_for**. In Figure 43, the rule

check_for(sub_cover,tar_paper,[tar_paper1,tar_paper2,tar_paper3])

is used to verify that all sub-covers made out of tar-paper use tar-paper1, tar-paper2 or tar-paper3. In addition, those types of tar-paper not used are listed as possible material substitutions. These lists of possible substitutions will become important again when determining raw material requirements later in the processing. Figure 44 illustrates the use of this type of standards check.

```
check for sub_cover sub_cover14

    sub_cover subcover14 meets requirements; allowed substitutes are:

      - tar_paper1

      - tar_paper3
```

Figure 44. Substitution of Materials

Note that the design data currently has sub-cover14 made from tar-paper2. The other two types are listed as possible substitutes. For more realistic situations, substitutions of one material may affect other parts of the product. For example, consider a case where several types of plastic have been listed as acceptable for the product piece in question. If glue is being used on the plastic during the manufacturing process, different plastics may require different adhesives. Therefore, caution must be used in making substitutions. Sample output from the standards checking portion of the translator is shown in Figure 45.

```
check for frame frame3

    grade marks must be clearly visible on all framing
    members for inspection

check for sub_cover sub_cover7

check for cover cover6

    approved methods must be used for building masonry walls
    when outside air temperature drops below 40 degrees
    fahrenheit
```

Figure 45. Sample Translator Output (Simulated)

(2) Product Assembly

Once the standards checks have been completed, the translator must determine the product assembly sequence. To build our one room house, we would expect the foundation to be erected first. Figure 46 is a listing of Prolog rules used to generate the assembly steps for the frame foundation and walls.

The first frame selected for assembly is the foundation. This frame is located by finding a face which is part of the house being built and which also faces away from the ground. The **trans_partof(Yface,H)** will locate any face that is part of the house represented by the variable H. Then **normal_Z(Yface,1)** checks if the Z component of the normal to the face of interest is equal to one. If so, then this face is a floor. Figure 47 shows example orientations of normals for our one room house.

Any normal parallel to a coordinate axis will have that axis' component equal to one in value if it points in the positive direction along the axis and equal to minus one if it points in the negative direction. For the example house, only the normals to the faces contained in the roof do not meet these requirements. It is not

```
/* do foundation frame */
assemble (H, house) :-
    kind_of (Yface, face),
    trans_partof (Yface, H),
    normal_Z (Yface, 1),
    contains (Yface, L),
    member (Frame, L),
    kind_of (Frame, L),
    property (Frame, material_type, Mtype),
    assertz (operation (Frame, assemble, 'material type:', Mtype)).

/* do frame perpendicular to ground */
assemble (H, house) :-
    kind_of (Yface, face),
    trans_partof (Yface, H),
    normal_Y (Yface, 0),
    normal_Z (Yface, 0),
    contains (Yface, L),
    member (frame, L),
    kind_of (Frame, frame),
    property (Frame, material_type, Mtype),
    assertz (operation (Frame, assemble, 'material type:', Mtype)).

assemble (H, house) :-
    kind_of (Yface, face),
    trans_partof (Yface, H),
    normal_X (Yface, 0),
    normal_Z (Yface, 0),
    contains (Yface, L),
    member (Frame, L),
    kind_of (Frame, frame),
    property (Frame, material_type, Mtype),
    assertz (operation (Frame, assemble, 'material type:', Mtype)).
```

Figure 46. Frame Assembly Rules

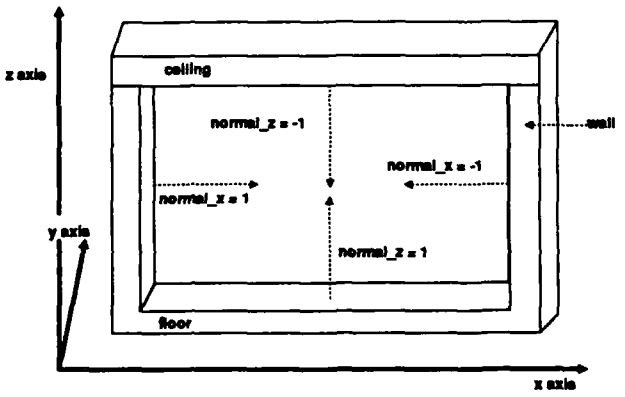


Figure 47. Orientation of Normals

necessary that any face meet this requirement; it has been done only to simplify the example.

Once the floor frame is in place, the second and third rules in Figure 46 locate the wall frames and add them to the assembly list. The second rule looks for faces with normals parallel to the X axis by specifying that the Y and Z components of the normal are equal to zero. Similarly, the third rule locates those faces parallel to the Y axis. In Prolog, backtracking will force these rules to be tried until no more valid solutions are found. In this way, we locate all faces meeting the specifications of each rule. Therefore, we only need be sure that each rule does indeed fully state all specifications of concern.

In Figure 48, the rules which generate assembly data for the ceiling and roof are shown. The only notable difference from our previous rules in Figure 46 is that faces associated with the roof are located by using the contains relation associated with the roof. This is a better method than using normals since the normal vector for a roof face can vary depending on the design of the house. The only framing left to be performed is for the windows and doors. Figure 49 lists the rules which handle these two cases.

```
/* ceiling frame */
assemble (H, house) :-
    kind_of (Yface, face),
    trans_partof (Yface, H),
    normal_Z (Yface, -1),
    contains (Yface, L),
    member (Frame, L),
    kind_of (Frame, frame),
    property (Frame, material_type, Mtype),
    assertz (operation (Frame, assemble, 'material type:', Mtype)).

/* roof frame */
assemble (H, house) :-
    kind_of (Roof, roof),
    trans_partof (Roof, H),
    kind_of (Yface, face),
    trans_partof (Yface, Roof),
    contains (Yface, L),
    member (Frame, L),
    kind_of (Frame, frame),
    property (Frame, material_type, Mtype),
    assertz (operation (Frame, assemble, 'material type:', Mtype)).
```

Figure 48. Assembly Rules

```
assemble (H, house) :-
    kind_of (Door, door),
    trans_partof (Door, H),
    property (Door, material_type, Mtype),
    assertz (operation (Door, assemble, 'material type:', Mtype)),
    get_faces (Door, Face1, Face2),
    assertz (operation (' ', '-attach to', Face1, Face2)).

assemble (H, house) :-
    kind_of (W, window),
    trans_partof (W,H),
    contains (W,L),
    member (Sill, L),
    kind_of (Sill, sill),
    assertz (operation (Sill, assemble, 'window sill for:', W)),
    get_faces (W, Face1, Face2),
    assertz (operation (' ', '-attach to:', Face1, Face2)).
```

Figure 49. Rules for Framing

Again, both rules only check particular parts of the house. For the door, we determine its material and the two faces to which it is attached. The same is done for the window except that the sill is treated as its frame. Again, backtracking is used to get all occurrences of windows and doors.

With all the framing in place, the faces must now be constructed. Figure 50 gives the code to handle this. Note that the exterior and roof are constructed first, and then the interior room itself. For each area, the contains relation is used to get a list of all components, including faces, and the information is passed to an **assemble(L,face)** routine to erect only the faces. This is actually a series of routines that use both backtracking and recursion to determine the assembly data. Figure 51 gives the routines that start the process.

The first and second rules in the list handle two different cases, non-floors and floors respectively. Any face pointing directly upward is considered a floor. In our example, there is only one floor. The first rule takes precedence over the

```
assemble (H, house) :-
    kind_of (E, exterior),
    trans_partof (E, H),
    contains (E, L),
    assemble (L, face).

assemble (H, house) :-
    kind_of (R, roof),
    trans_partof (R, H),
    contains (R, L),
    assemble (L, face).

assemble (H, house) :-
    kind_of (R, room),
    trans_partof (R, H),
    contains (R, L),
    assemble (L, face).
```

Figure 50. Rules for Assembling Faces

```
assemble (L, face) :-
    assemble1 (L, [], face).

assemble (L, face) :-
    member (Face, L),
    normal_Z (Face, 1),
    assertz(operation (comment, 'build floor as last step', _, _)),
    contains (face, L1),
    assemble2 ([L1], [L1], face).

assemble1 (L, L1, face) :-
    member (Face, L),
    not (Normal_Z (Face, 1)),
    delete (Face, L, L2),
    contains (Face, L3),
    assemble1 (L2, [L3|L1], face), !.

assemble1 (L, L1, face) :-
    assemble2 (L1, L1, face), !.
```

Figure 51. Rules for Components of Faces

second and calls the third rule in the same figure. The third rule simply finds all faces which are part of the area of concern but are not facing upward. Looking at the left side of the third rule, **assemble1(L,L1,face)**, L is the set of parts determined using the contains relationship earlier and L1 is a set which we will construct. L1 is initialized to nil when the first rule calls the third rule. When the third rule finds a face meeting its requirements, the contains relation is again used to determine the parts of the face. This set of parts is added to L1 and assemble1 recursively calls itself looking for more faces. When none are found, we fall through to the fourth rule which calls assemble2. The ! symbol at the end of the assemble1 rules prevents the system from backtracking into them. It will only proceed forward into these rules. Backtracking is not necessary since we exit these rules only when all faces meeting our specifications are found.

Looking again at the second rule in Figure 51, we put only one face in the list at a time. Backtracking is necessary in the case where there is more than one possible floor face. This may or may not be desirable depending on the house design. For the other faces, a list of all faces in the area of concern is created using recursion to allow a search for common building materials to better organize the design data.

Figure 52 shows the routines necessary to complete the face assemblies. Note that assemble2 will recursively call itself until there are no face parts left. It then falls through to the last rule which succeeds and thus exits. Again, no backtracking is allowed or necessary.

The first two rules in Figure 52 search for all sub-covers letting those sub-covers made of material already used in the area of concern take priority over material not yet used. This is accomplished by searching through all the current operation predicates looking for sub-covers already processed that use the same material. If such a sub-cover is found, then a search is performed over the list of all sub-covers in the area of concern to attempt a match. If a match is found, then that material has already been used and will take priority. If no match is found, then the next sub-cover in the next face is listed in the assembly report.

```
assemble2 (Full_L, L, face) :-
    member (Face, L),
    delete (Face, L, L1),
    member (Item, Face),
    kind_of (item, sub_cover),
    property (item, material_type, Mtype),
    operation (Y, _, _, Mtype),
    member (Face1, Full_L),
    member (Y, Face1),
    assertz (operation (item, assemble, 'material type:', Mtype)),
    assemble2 (Full_L, [Face2|L1], face), !.


assemble2 (Full_L, L, face) :-
    member (Face, L),
    delete (Face, L, L1),
    member (Item, Face),
    kind_of (item, sub_cover),
    property (item, material_type, Mtype),
    assertz (operation (item, assemble, 'material type:', Mtype)),
    delete (item, Face, Face1),
    assemble2 (Full_L, [Face2|L1], face), !.


assemble2 (Full_L, L, face) :-
    member (Face, L),
    delete (Face, L, L1),
    member (Item, Face),
    kind_of (item, cover),
    property (item, material_type, Mtype),
    not (liquid ( (Mtype, paint, _, _, _, _))),
    operation (Y, _, _, Mtype),
    member (Face1, Full_L),
    member (Y, Face1),
    assertz (operation (item, assemble, 'material type:', Mtype)),
    delete (item, Face, Face2),
    assemble2 (Full_L, [Face2|L1], face), !.


assemble2 (Full_L, L, face).
```

Figure 52. Rules for Completing Faces

The third and fourth rules provide a similar function for the covers except that covers made from paint are not yet allowed to be listed. The painting will be done at the end of the house construction to prevent damage to the finish.

The house is now close to completion. The window panes are inserted into place, the windows and doors are painted, and the doors are installed using the appropriate doorknobs and hinges. Now is the time to complete the painting of the faces which was previously postponed. Figure 53 shows the rules for painting faces.

```
assemble (H, house) :-
    kind_of (R, roof),
    trans_partof (R, H),
    contains (R, L),
    paint_face (L).

assemble (H, house) :-
    kind_of (E, exterior),
    trans_partof (E, H),
    contains (E, L),
    paint_face (L).

assemble (H, house) :-
    kind_of (R, room),
    trans_partof (R, H),
    contains (R, L),
    paint_face (L).
```

Figure 53. Rules for Painting Faces

Note that first the roof is painted (if necessary), then the exterior and lastly we paint any interior surfaces. The **paint_face** routines are similar to ones we have previously discussed. The one room house is now fully constructed.

(3) Raw Materials Listing

With the assembly data finished, the translator must now determine the raw material requirements to build the house. This is done by calling on the

raw_materials_needed rules. All the rules work in much the same manner. They first determine what component is being considered, then the material associated with this component, and lastly the dimensions of the component. All dimensions are converted to a common unit of measurement prior to calculations. Those parts of the house associated with a face such as a cover or sub-cover call a routine **get_area** to determine the surface area involved. This special routine is necessary since faces may have areas such as doors, windows and openings which subtract from the total area of the face to be covered. This is handled by calculating a negative area for each face to be subtracted out prior to material requirements calculations. This negative area is then asserted as a fact for each face prior to actual entry into the **raw_materials_needed** routines. A sample calculation routine is shown in Figure 54.

```
raw_materials_needed :-
    kind_of (Extens, sub_cover),
    dimension (Extens, depth, Th, Thunits),
    property (Extens, material_type, Material),
    material (Material, _, Ht, Htunits, Wd, Wdunits, Dp, Dpunits,
        _, _, _, _, Cost),
    match (Ht, Htunits, Wd, Wdunits, Dp, Dpunits, Th, Thunits,
        Act_Ht, Units1, Act_Wd, Units2),
    get_area (Extens, Area, Units),
    convert (Act_Ht, Units1, Act_Ht2, Units),
    convert (Act_Wd, Units2, Act_Wd2, Units),
    Num_Units is (Area / Act_Ht2 * Act_Wd2)),
    Tot_Cost is (Num_Units * Cost),
    add_material (Material, Num_Units, Tot_Cost), fail.
```

Figure 54. Sample Calculation Routine

One aspect of how the above example works not yet mentioned is the call to **match**. This rule attempts to find a match between the dimensions of the material to be used and the thickness of the sub-cover within an acceptable tolerance. This information is then used to determine the orientation of the material within the

sub-cover. For example, if a board measuring two inches by four inches by four feet is used to build a sub-cover which is four inches thick, then the two inch dimension would be used for area calculations. This type of check is necessary since the dimensions height, width, and depth are based on the view of the person determining the values.

Once the units of material required and cost are determined, these values are added to the total by calling **add_material**. This rule first checks for any previous data on this material. If some is found, then a new total is calculated and saved. Otherwise, a new fact on the material of concern is created and saved.

The only other unusual calculation performed during the material calculations determines the frame requirements along the center of the roof, between the roof and the ceiling. We need the height of the roof above the ceiling to make this calculation. This is easy to do since the normal vectors for the roof faces are known. It turns out that each component of the normal is equal to the cosine of the angle created by the intersection of a line parallel to that component's axis and the plane containing the other two axis [Ref. 105]. Figure 55 demonstrates this concept.

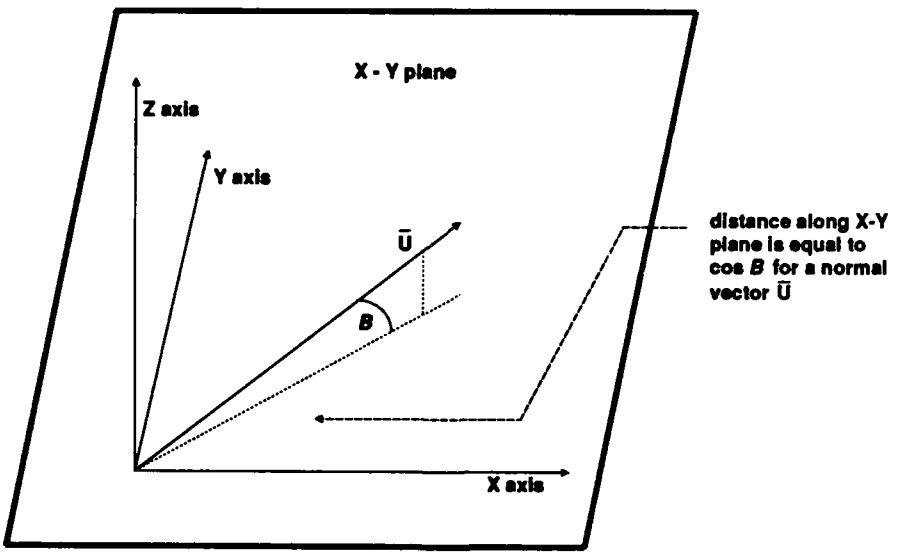


Figure 55. Computation of Normal

The Z component of the normal vector is equal to the cosine of the angle created by the intersection of the normal and the plane containing the X and Y axis. With this fact, we can calculate the angle of intersection, Beta, of the roof and the house. Using the dimensions of the roof faces, it is now possible to determine the height of the roof above the ceiling since sin(Beta) is equal to the height of the roof above the ceiling divided by the length of the roof face.

Once the quantities of materials and their costs have been determined, a Raw Materials Report is produced. The report lists the units required and cost for each raw material used. Following the list of raw materials is a total cost for the product. Figure 56 gives an example of this report.

After the initial Raw Materials Report, the translator examines possible material substitutions reported during the standards checks and makes each substitution, one at a time, to generate a new report. Figure 57 is an example of a modified Raw Materials Report output by the translator. It shows the cost for parts when sub-cover14 is made out of tar-paper1 instead of tar-paper2.

D. SUMMARY

In this chapter we described the high-level approach to integration in detail. We presented the data requirements for integrating CAD and CAM, the two activities interfaced by our expert system translator. The data interactions between these two activities were described in detail and demonstrated in our implementation of the translator.

Raw Material Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.5
wood8	$3582	434.2
tar_paper2	$841	6.7
hardboard32	$211	1.5
hardboard34	$147	1.5
hardboard78	$200	.7
hard_wood9	$900	75
sheath_paper24	$64	.9
shingle12	$2020	1616
brick88	$4224	3673
paint9	$8	1
paint17	$4	.6
paint21	$12	.9

Total material cost is $13996

Figure 56. Raw Materials Report

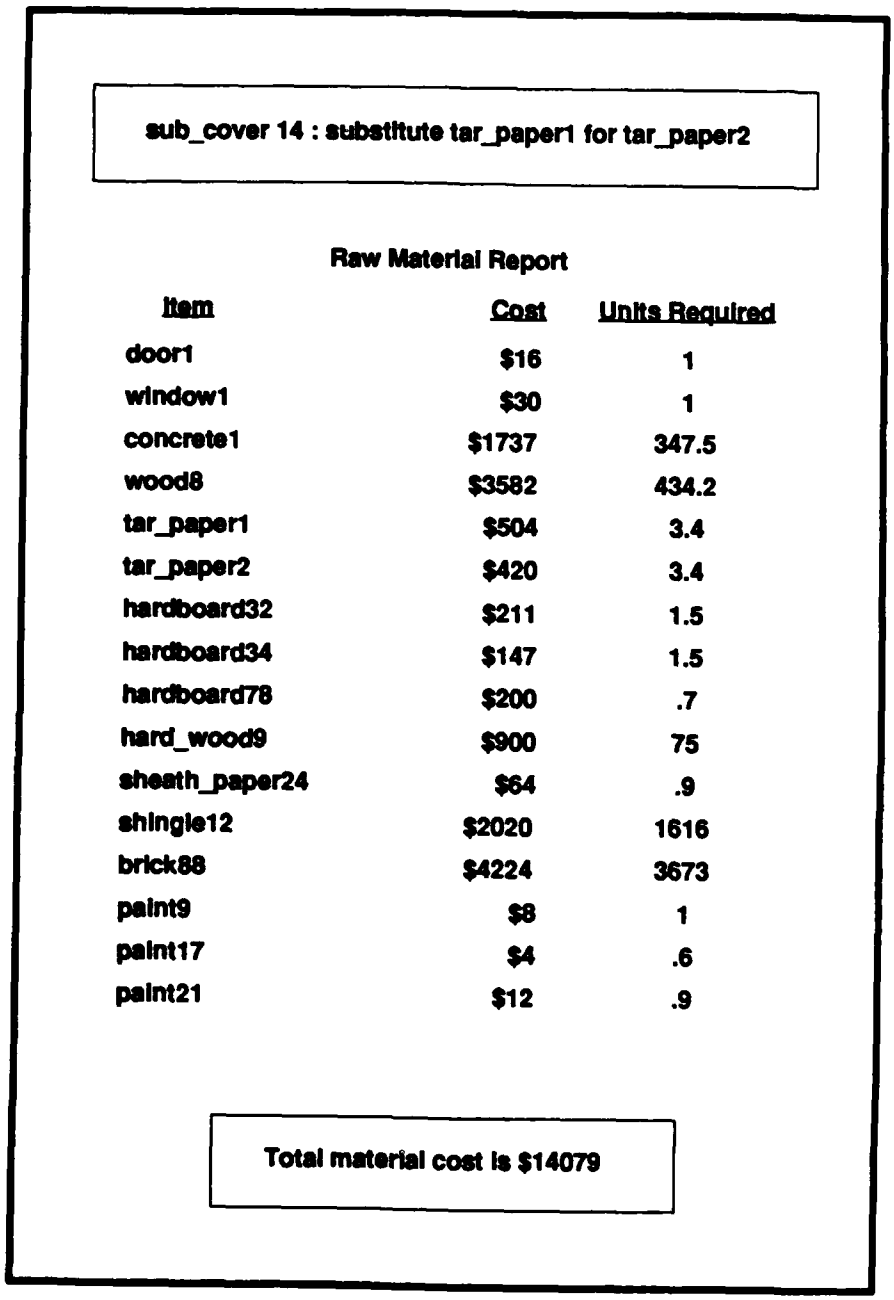


Figure 57. Modified Raw Materials Report

VII. LOW-LEVEL INTEGRATION OF MANUFACTURING FUNCTIONS

A. MOTIVATION

In Chapter VI, we discussed the integration of product design and production functions using an expert system translator. We discussed the manufacturing cycle and the role that design functions play in that cycle by describing their data interactions. Unlike previous process-oriented approaches, where a simple interface is placed between manufacturing processes, our overall approach can be characterized as data-oriented. With our approach, manufacturing activities will be grouped into several cooperating systems, each with a single database server as its core; an approach we call low-level integration.

In the current manufacturing environment, islands of automation provide computer support for most manufacturing activities. In other words, there is a separate computer system for each of the boxes shown in Figure 58. Some automate the design drawing process, others automate process planning functions, and still others automate the ordering of inventory parts. Although each phase is now more or less automated, the full potential of computer-supported manufacturing cannot be realized unless these diverse computer systems communicate adequately with each other. In the current environment, data used in one phase cannot be used directly by the system supporting another phase. For example, design data of a product in a computer-aided design (CAD) system cannot be used directly by a computer-aided manufacturing (CAM) system for process planning because these systems use completely different formats for data storage. To make things worse, these data formats are often proprietary.

So the industrial and mechanical engineers have asked themselves an inevitable question "Is it possible to have Computer Integrated Manufacturing(CIM), that allows all phases of manufacturing to utilize each other's data?" Many papers on the Computer

Integrated Manufacturing topic which we have come across in the proceedings of conferences such as AUTOFACT and CIM-International use the high-level approach discussed in Chapter VI. At the early stage of our research, we too wrote a Prolog data translator, also described in Chapter VI. The high-level approach is nice in that already existing systems need not be modified. But the approach is not a long term solution to the integration problem. This is merely an interface, a bridge shall we say, that simply "connects" different components. The term "integration" should mean an embodiment of pieces into a working whole, not just a juxtapositioning of them. By adding the translators into the system, the whole manufacturing environment becomes even more complex; there are more pieces of software to take care of. What would happen if the internal workings of one component system are modified? We must write a whole new

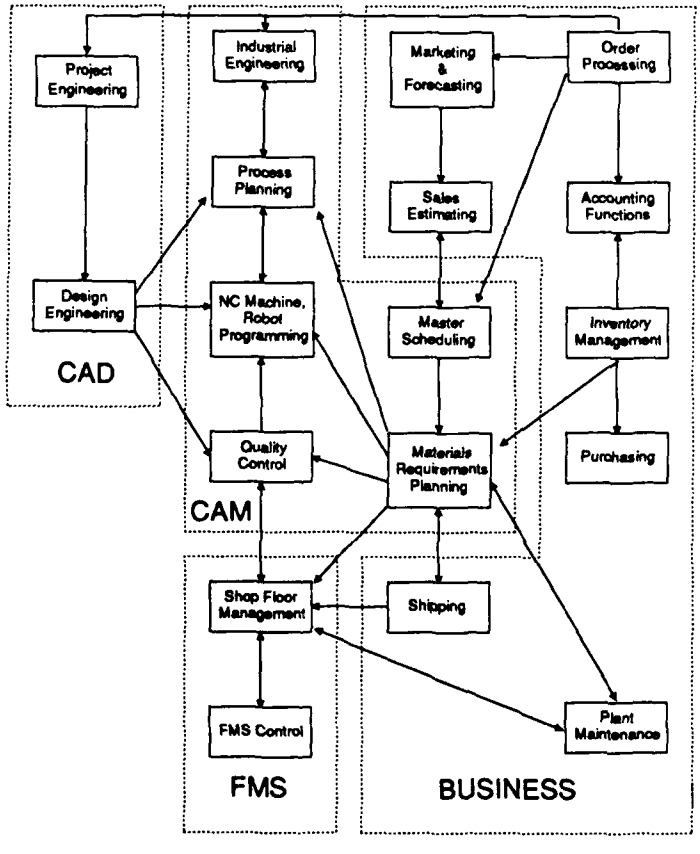


Figure 58. Manufacturing Functions

translator! This approach may be acceptable as a short-term, interim solution, but not as a final solution for true integration.

There is a second approach. The Society of Manufacturing Engineers (SME) proposed the CIM Enterprise Wheel approach depicted in Figure 59. As we can see from the figure, the critical part is a common data server, the kernel of the Enterprise Wheel. The figure makes sense. But, is it realizable?

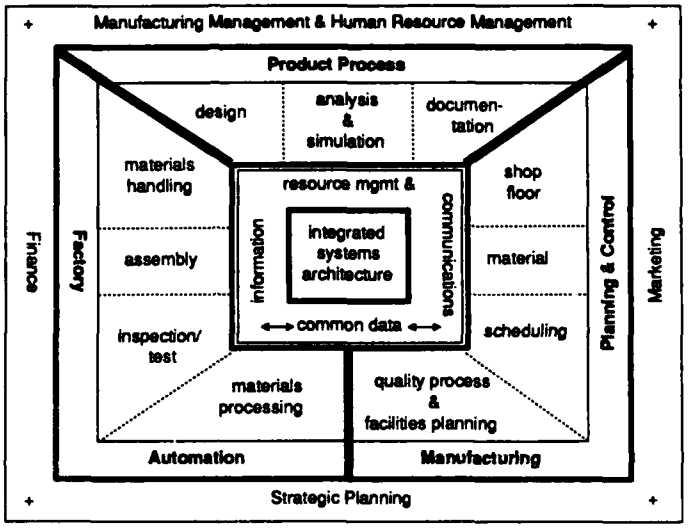


Figure 59. CIM Enterprise Model

There are actually two ways to view this figure. The first one is to literally treat the kernel as a centralized data management system. This centralized data server is normally interpreted as a relational database management system. When the relational database management system came into existence it was adapted in many diverse application areas beyond the obvious business data processing. One of them is in the manufacturing environment. We in the database community generally agree that for the relational DBMS to be truly applicable to non-traditional areas, it must be extended to handle more complex objects and semantics involved in manufacturing environment. Many good

projects, such as [Refs. 100, 103, 109, 110, 111] and others, have focused on handling complex objects and semantics in the database community. But many papers we have come across in the proceedings of industrial and mechanical engineering conferences such as AUTOFACT use the unextended, currently available relational systems. They have treated an unextended relational system as a panacea for the data handling problem in Computer Integrated Manufacturing. We in the database community know better: either the relational system must be improved or a completely new system must be developed.

The second way to view Figure 59 is that the kernel data server is not the physical, centralized database but a logical one. The kernel is a common data model applicable to all phases of manufacturing. The actual operating environment will realize this kernel as a collection of distributed, semi-autonomous database systems. In our research, we have taken this approach -- a common data model applicable to all phases of manufacturing -- with one major difference from what is envisioned in the CIM Enterprise Wheel.

After reviewing other works in Computer Integrated Manufacturing, we noticed a remarkable commonality among them. Whether they use the interface or Enterprise Wheel approach, they are all process-oriented. They accepted the traditional way of categorizing manufacturing activities into design, engineering, numerical control programming, process planning, inventory control, scheduling, etc. and then proceeded to "integrate" them. On the other hand, our work may be classified as data-oriented. We ignored the traditional categorization of activities.

We will show that the single activity of designing a product also outputs the product's process plan. In other words, from a data-oriented perspective, design and process planning are not separate phases of manufacturing because both of them can be supported by a single data server. We will rename them as the *preparatory phase* of manufacturing.

We will also describe how our proposed data model, without any modification, can be used to capture the semantics involved in describing a shop floor layout. This will be the main information used in the Production Monitoring stage shown in Figure 1. The

function which requires this information is the scheduler. We will describe a simulation technique for scheduling and show that data for simulation is readily available from a database server maintaining a shop floor layout -- a direct benefit of a data-oriented approach. Consider what will happen if a process-oriented approach is taken for the scheduling problem. In other words, what would we do, given a task of automating a scheduling problem? We would first propose some kind of algorithm for finding an optimal or near-optimal solution. Then, we would identify the data required as input to the algorithm. Since we have solved the problem without much regard to the data already in some database, we most likely will not be able to find a single source for all required data. We will thus create a special data server, which holds all necessary data, just for the scheduler. This data server will necessarily hold duplicate information, which leads to potentially harmful data inconsistency. Since this data server is not connected to other data servers in any way, data must be extracted manually. It is not economical to do manually, so eventually we will develop software to automatically extract data from other data servers. This software is exactly the translator we mentioned in the high-level interface approach to integration. It should be clear from this example that this approach is in fact exacerbating the integration problem.

Perhaps our major contribution is breaking the prevalent "mind set" of the process-oriented approach. We hope that once this "mind set" is eliminated, expedient progress toward true integration can be made.

B. THE DATA-ORIENTED APPROACH

In our preliminary research, we divided the basic manufacturing functions and activities into four stages based on the type of function they performed. These four stages were depicted in Figure 1, which is reproduced as Figure 60 for convenience. We applied our data model to the design stage and developed the high-level translator described in Chapter VI [Ref. 94]. After further research, we published a comparison of the three approaches to integration discussed in Chapter III [Ref. 112]. At this point we decided that the path to true integration involved the low-level integration approach, and

conceptualized our data-oriented solution [Ref. 113]. We applied our data model to the process planning function, the focal point of the production planning stage of Figure 60, and recognized the relationship that process planning has to product design using our data-oriented perspective [Ref. 114]. We then determined that application of the same data model to shop floor layout, the main data manipulator in the production monitoring stage, would integrate the product design, production planning, and production monitoring stages [Ref. 115]. The fourth stage, business activities, could be easily integrated with the other three since the information used in business activities is available as a byproduct of the other three stages.

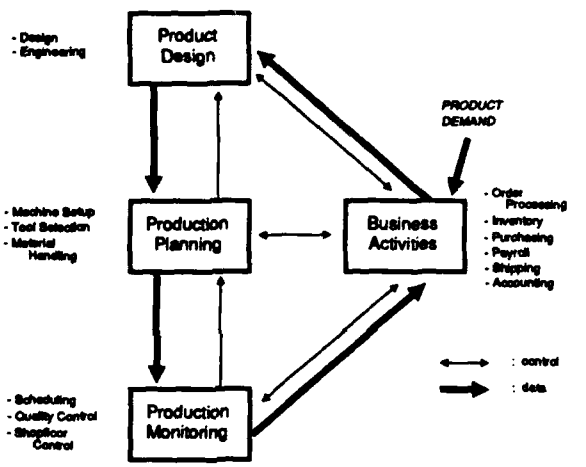


Figure 60. Stages of Manufacturing

In this chapter, we will discuss our data-oriented approach in more detail. We will apply our data model to product design and process planning, which we have combined to form the preparatory phase of manufacturing. We will conclude the chapter with the application of our data model to production monitoring.

1. Preparatory Phase of Manufacturing

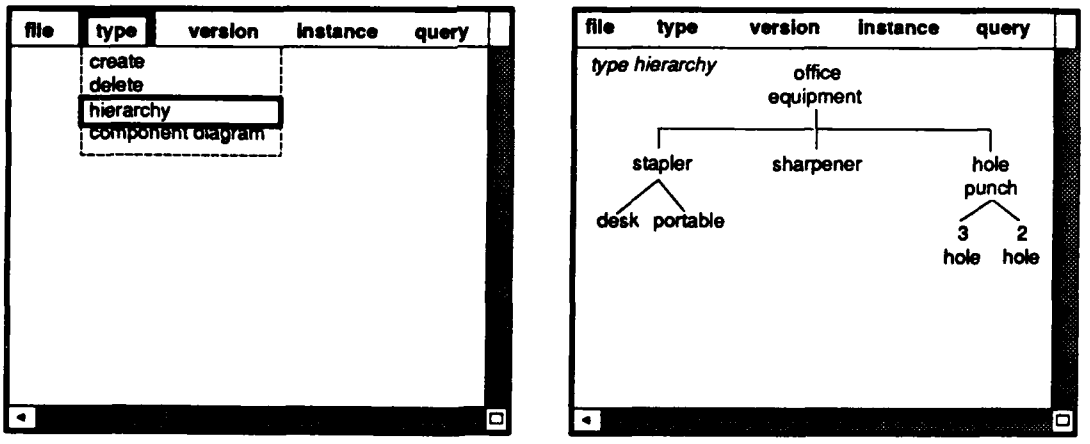
We will begin this section by showing how our data model supports the semantics of product design. We will then discuss our approach to process planning and

the application of our data model to it. Our discussion continues with the integration of the design and process planning functions.

a. Modeling the Semantics of Product Design

There are a number of concepts inherent in product design which need to be modeled in order to adequately support that environment. Perhaps the best way to demonstrate the support provided by our model is to describe the design process from the viewpoint of the design engineer, showing at each step of the process how the model handles the semantics involved. The explanation will be accompanied by figures depicting a simulated user interface to our model.

The first decision faced by a design engineer in designing a new product is choosing the best starting point. The new product can be either designed from scratch or designed by modifying a previous product design. If the new product is designed from scratch, a type hierarchy will be shown to the engineer. The types in this hierarchy will be related by the generalization and specialization abstraction concepts. There will be one type in the hierarchy for each product in the application domain. Figure 61(b) shows a



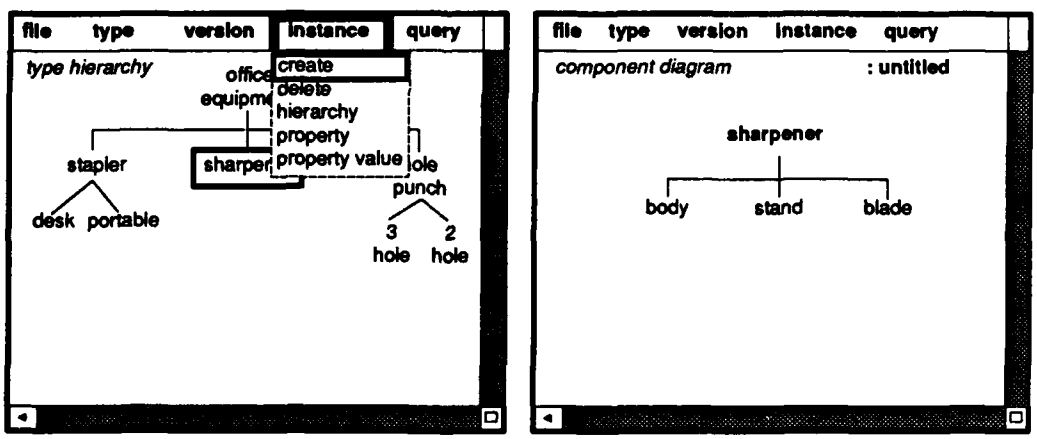
(a) command issued

(b) result

Figure 61. Type Hierarchy Displayed

sample type hierarchy used in the design of office equipment. This hierarchy was produced as a result of the engineer selecting the **type** menu and the **hierarchy** command, as shown in Figure 61(a). The design engineer will select a type from the hierarchy by positioning the arrow cursor (not shown in the figures) on the desired type and pressing the left mouse button. This is indicated graphically by the bold rectangle surrounding the selection.

Once a type is chosen, an instance of that type is created and a component diagram will be made available for further manipulation. The component diagram consists of other types related to the chosen type by the aggregation abstraction concept. The instance of a selected type is created by using the **instance** menu and **create** command, as shown in Figure 62(a). Using a combination of the information about the type selected and the menu and command choices, the appropriate component diagram will be displayed (Figure 62(b)).



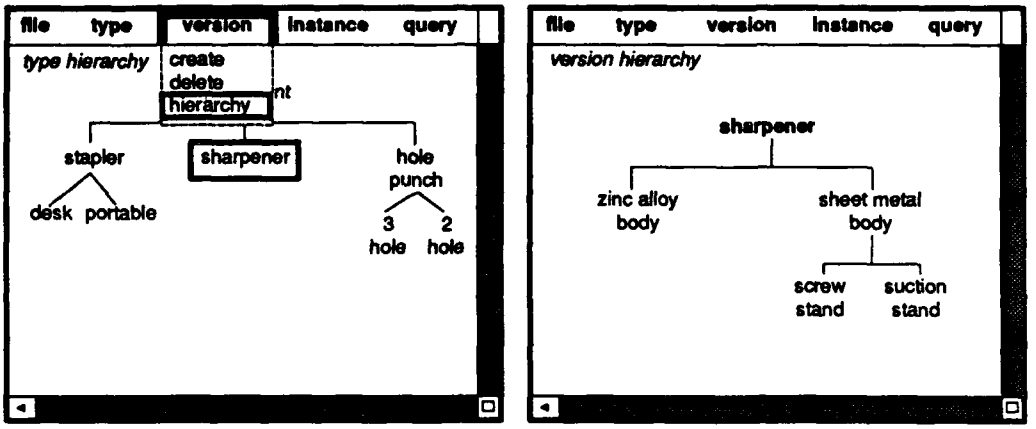
(a) command issued

(b) result

Figure 62. Instance of a Type Created

If the new product is designed by modifying a previous product design, the engineer selects the appropriate type from the type hierarchy and asks for the version

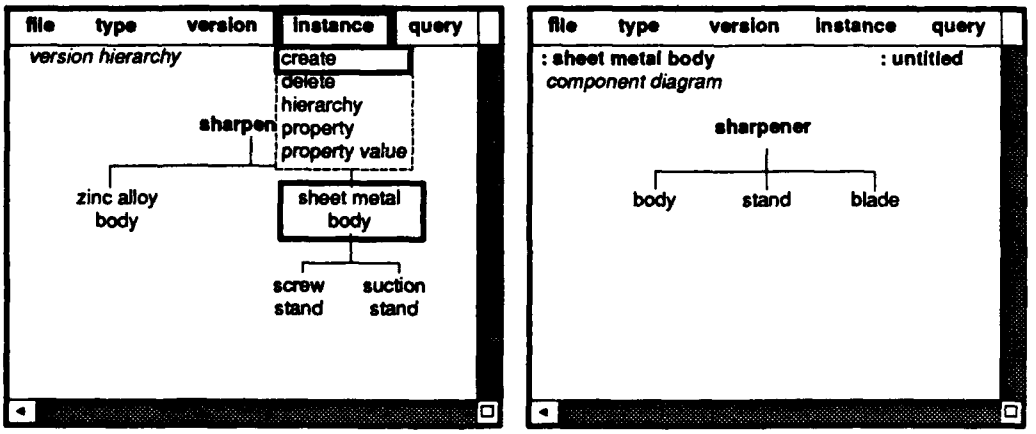
hierarchy for that selection (Figure 63). The engineer selects the version from the version hierarchy which is the closest to the new product being designed. An instance of the selected version is created. The component diagram for the type from which the selected version was created is then made available to the engineer (Figure 64).



(a) command issued

(b) result

Figure 63. Version Hierarchy Displayed

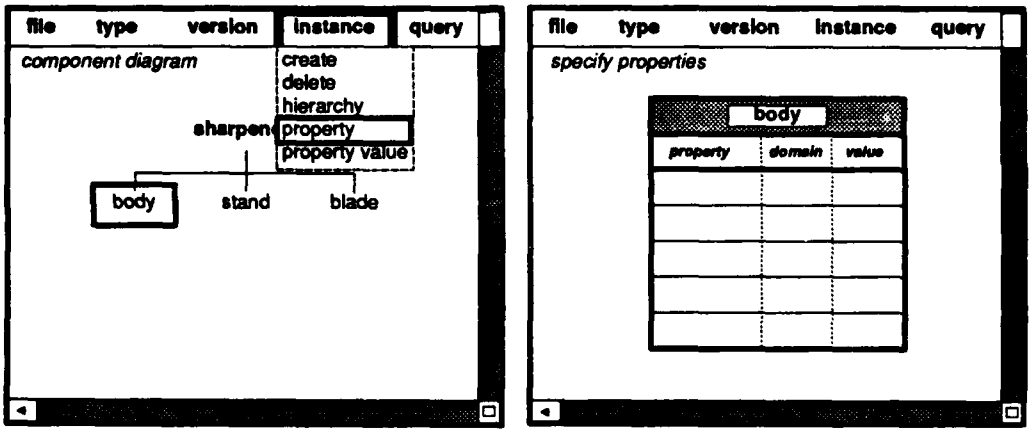


(a) command issued

(b) result

Figure 64. Creating an Instance of a Version

At this point the engineer has an instance of either a type or an instance of a version and a component diagram to work with. We will continue the discussion assuming that the engineer is working with an instance of a type. As the property values for the instance are provided and new component types are chosen from the component diagram, the design schema for the new product takes form. Figure 65(a) shows a selected component of the component diagram and the **property** command in the **instance** menu being invoked. A prototype for the selected component appears, as shown in Figure 65(b), with slots for property names and values. The property name, its domain, and its value can be specified at this time, or the engineer can defer the property value specification until a later time. If the property values are specified separately, the appropriate component of the component diagram is selected and the **property value** command in the **instance** menu is issued, as shown in Figure 66.



(a) command issued

(b) result

Figure 65. Specifying Properties

A distinction should be made between the two forms of aggregation found in Figure 65. The component diagram shows an aggregation of the types which make up a compound object (Figure 65(a)). The prototype is an aggregation of the properties of an

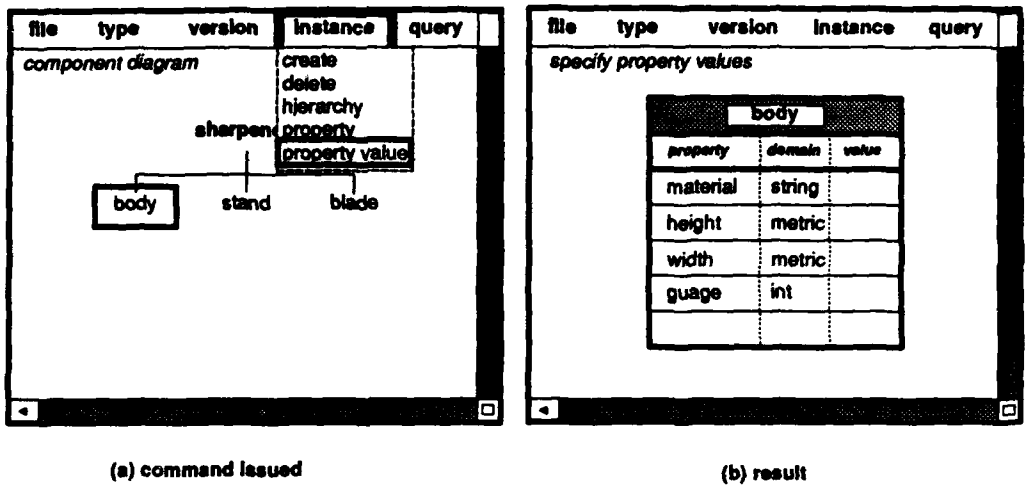


Figure 66. Specifying Property Values

object. The component diagram represents the structural content of an object while the properties in the prototype represent the informational content of the object they relate to.

The engineer continues defining properties and/or specifying property values until either a point is reached in the design where multiple alternatives are necessary, or the design is complete. If multiple alternatives are desired, the engineer selects the command to create an instance hierarchy. In this process, a copy of the current instance, with its properties and property values, will be made for the new alternative. Figure 67(a) shows the **hierarchy** command in the **instance** menu being issued. This command works in several ways. First, if no instance hierarchy exists for the current design project, invocation of this command will create a new hierarchy. If a hierarchy already exists, invocation of this command will display it, which is the case shown in Figure 68.

For an instance hierarchy to be created, one or more property values from the current instance must be changed to create a new alternative instance. This can be done by either modifying an existing property value, e.g., **14** in Figure 67, or specifying a

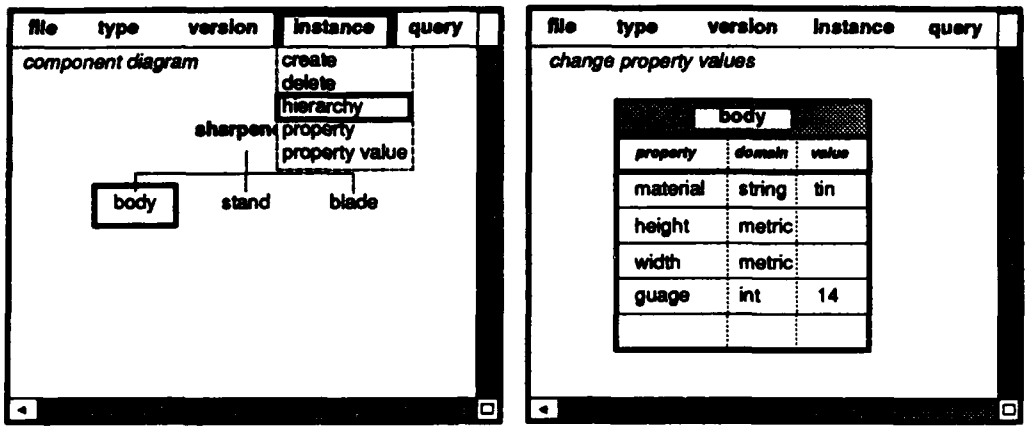


Figure 67. Creating an Instance Hierarchy

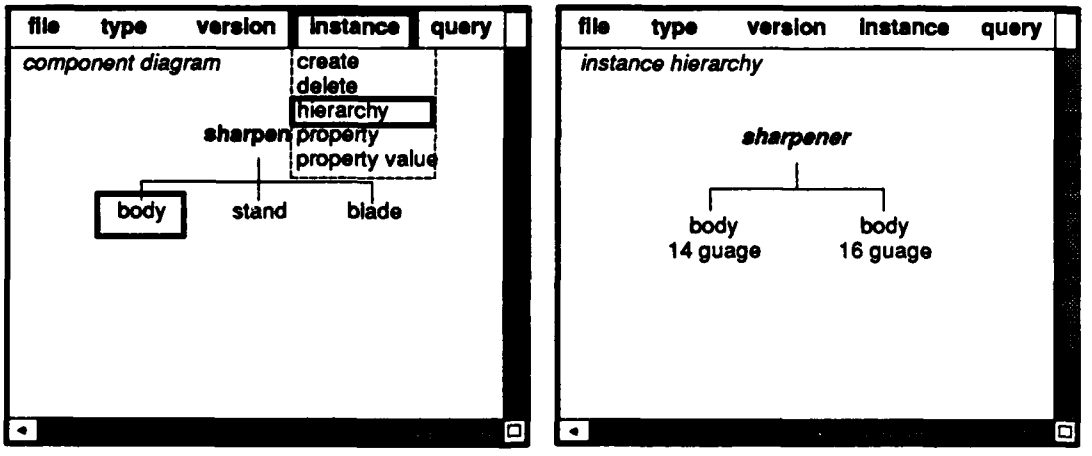
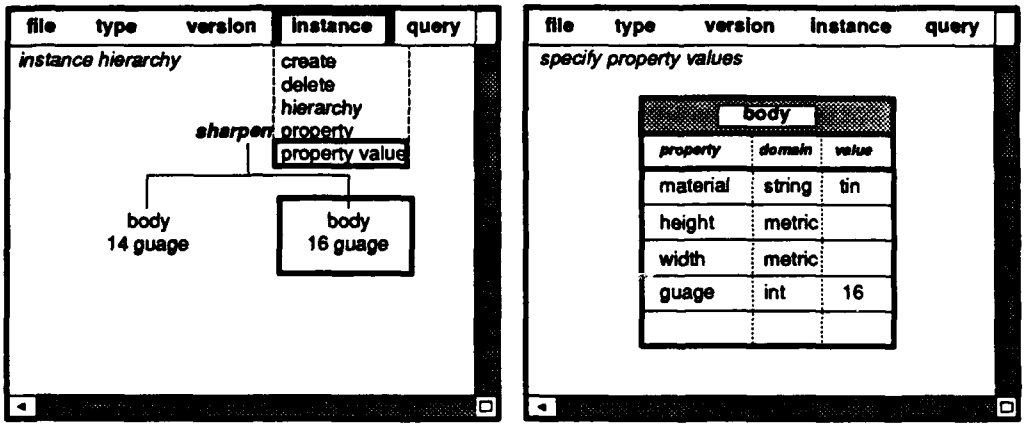


Figure 68. Displaying the Instance Hierarchy

value for a property whose value was previously undefined, e.g., **width** in the same figure.

To continue the design, the engineer will select one of the instances and use the component diagram to complete the design. Figure 69(a) shows the **body 16 guage** instance being selected and the **property value** command in the **instance** menu invoked. The result of this action, shown in Figure 69(b), is the resumption of the design process at the point where it left off before the instance hierarchy was created. When the design resumes, a prototype for the selected instance will be used.



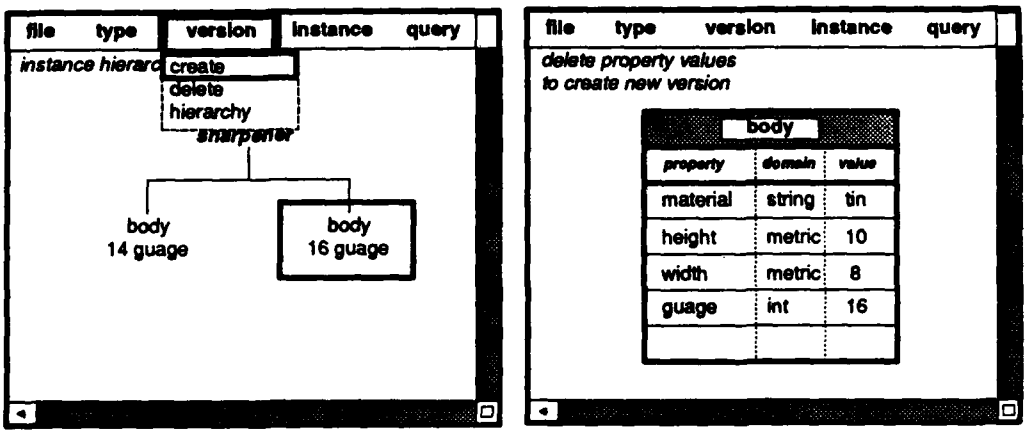
(a) command issued

(b) result

Figure 69. Continuing with Property Value Specification

When the design is complete, the engineer decides whether or not to archive the instance hierarchy since it will not be kept in its entirety by the system. The instance from the hierarchy which represents the final product design is associated with the version or type from which it was created using a command selected by the engineer. The engineer may also decide at this time to create a new version from the final product instance to be placed in the version hierarchy for the appropriate type. If a new version is desired, the engineer will determine which property values from the final product instance

will be used when the new version is created. Figure 70 depicts a new version being created from the **body 16 guage** instance which is now completely specified. Each of the prototypes which make up that instance will be presented to the design engineer so that property values can be deleted to create a new version. The new version will be placed in the version hierarchy using the **save** command in the **file** menu.



(a) command issued (b) result

Figure 70. Creating a New Version

It should be clear from the preceding description of the design process that our model provides the maximum possible flexibility at every step along the way. At the same time, the confusion to the design engineer about the data model operation is minimized due to the close correspondence between the modeling choices and the design process.

b. Our Approach to Process Planning

Process planning (PP) specifies the operations to be performed on different workpieces at different workstations in order to complete one production cycle. Instead of the traditional approach to PP, our data-oriented approach views PP as being divided into four phases. In the first phase, a gross decision on the process is made,

categorizing a product based on the degree of machining versus assembly in the process plan. The alternative decisions are whether a part should be 1) machined from raw material, 2) machined from a casting, or 3) assembled from smaller components. The first and second alternatives are forms of *parts manufacturing* where a workpiece is transformed from the unmachined state into the finished state by stepwise changes of its shape using machining processes. Assembly can be visualized as a process in which individual components such as parts and sub-assemblies are added to the finished product by assembly processes using assembly fixtures.

The second phase selects the appropriate operations and sequencing according to the decision made in the first phase. Each operation can be viewed as a transformation which takes a product from one state to another.

The third phase selects a machine type for each operation selected in the second phase. This selection is based on standard PP practice and does not take into account the actual availability of machines on the shop floor. In the parts manufacturing case, tasks performed during this phase include the layout of cutting sequences and a pattern, determination of cutting parameters, i.e., depth of cut, feed rate, speed of cut, and calculation of machining times.

The fourth phase selects a tool type for each machine type selected in the third phase. Again, this selection is made based on standard PP practice and not on availability of specific tools.

c. Modeling the Semantics of Process Planning

In the traditional manual process planning activity, data describing a product is placed on paper in the form of drawings and specifications. Both are revised and developed to higher levels of detail, potentially producing redundant and sometimes incomplete data. The redundant data leads to maintenance and consistency problems. The data which is produced in one process plan has little chance of being used in subsequent plans due to its manual nature. The engineering drawings produced during this manual process are given to a planning engineer who decides which operations,

machines, and tools are required. We will show how the use of our model in process planning alleviates these problems and provides a natural environment for an industrial engineer to work in.

Our approach to PP is to represent the set of alternative process plans for a product family as an acyclic directed graph, with the possible choices from each phase of the PP activity present. Figure 71 provides an example of two alternative process plans for a pencil sharpener, represented as an acyclic directed graph. The middle portion of the graph is used by both alternatives to produce the blade for the sharpener. Given this example, we will describe the activities involved in producing the alternative process plans, again showing at each step how our data model handles the semantics involved.

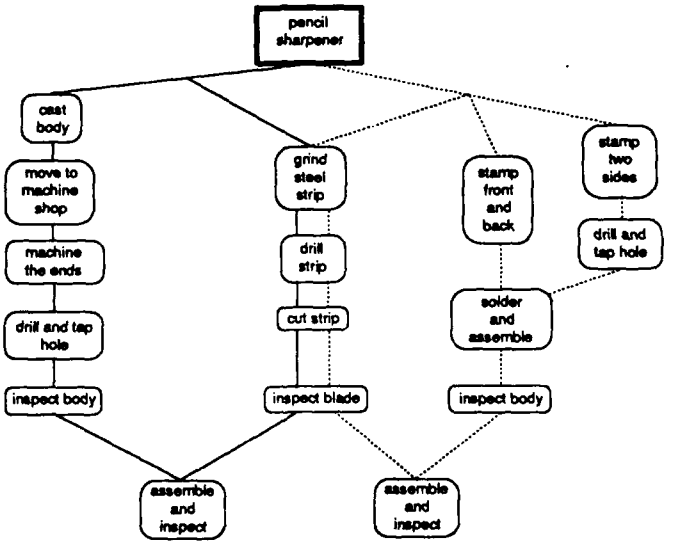


Figure 71. Alternative Process Plans

The industrial engineer has to be familiar with the product to be manufactured before beginning process planning work. Using our paradigm, the same conceptual schema used by the design engineer is available to the industrial engineer (as a component diagram). The conceptual schema represents a generic product to be designed

and manufactured. Using the information about the aggregations of types in the schema and the properties defined during the design process, the industrial engineer will develop a generic process plan.

Work on the process plan can be done from scratch or using previous work saved by the system, a situation analogous to that faced by the design engineer at the start of product design. If the decision is made to work from scratch, the industrial engineer will use the conceptual schema for the product to guide the development of the process plan. The first step is to create an instance of the type which the conceptual schema represents. Next, a component of the schema at the lowest level is chosen. The development of the process plan will be a bottom-up process, since the bottom-most portion of the conceptual schema represents the most primitive components of the product to be manufactured. Once the process plans for these primitives are defined, the next higher level can be considered. Since the levels in the conceptual schema are related by the aggregation abstraction concept, the process plan for each higher level will only have to deal with combining the process plans for the next lower level. Normally, this will entail some type of assembly procedure which is fairly easy to specify. The development continues, step by step, until the topmost level is reached.

For each primitive in the conceptual schema, the industrial engineer will determine which information can be specified directly and which has to be parameterized, or deferred. Parameters will be replaced by data for a specific product when this generic process plan is actually used in production. Information such as machine type and tool type may be specified during process plan development. However, other information such as the length of a cut will be a function of the dimensions of the workpiece and will therefore become a parameterized entry in the generic process plan.

If the industrial engineer chose to use previous work as a starting point, the version hierarchy for the type of product concerned would be displayed. As with the design engineer, a choice would be made form the hierarchy which is as similar as possible to the desired process plan. An instance of the selected version would be created

and work would proceed level by level, with the use of the conceptual schema, in the manner described previously.

When the generic process plan is completed, it will be added to the version hierarchy (in the appropriate place) of the type from which it was directly or indirectly created.

Once again, we have demonstrated that our model naturally supports a major manufacturing function. We have provided maximum flexibility in the development of process plans by making use of several of the abstraction concepts available in our model. By producing generic process plans which are parameterized and reusable, we have implemented the group technology concept and reduced the complexity of developing process plans.

d. Integrating Design and Process Planning Functions

We have shown the role that group technology plays in process planning and how our data model exploits that role in reducing the complexity of the problem of developing process plans. The abstraction concepts used in our data model easily capture the semantics of the process planning environment. As mentioned earlier, we have used the same data model to capture the semantics involved in the product design activities. We have been successful up to this point in applying the same modeling concepts to different manufacturing functions.

The design and process planning functions utilized the same conceptual schema and modeling concepts for their respective work. This means that for a given product, its conceptual design schema has a parameterized process plan associated with it. This parameterized process plan is a generic process plan for the family of parts represented by that schema. For example, Figure 72 is a conceptual design schema for a cabinet. The component **door** will have a generic process plan associated with it. The process plan is generic because the actual values for the plan are not yet specified. It is more like a template showing the general sequence of processes without the values of the parameters specified.

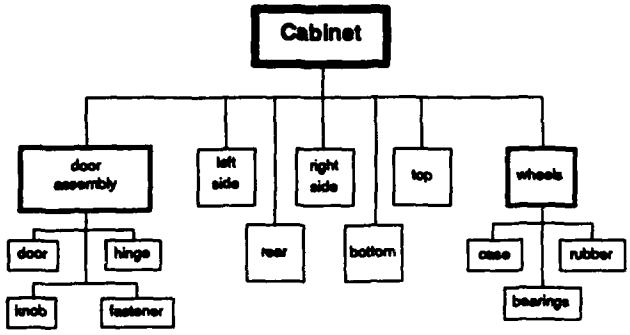


Figure 72. Conceptual Design Schema

When the designer creates a particular cabinet from this conceptual schema, he will fill in the design details such as the dimensions, color, type of door, etc. The generic process plan will simultaneously become a specific process plan with all the parameter values filled in for this particular cabinet. Thus, the design activity produces both the design and process plan for a particular product. There is no need for separate activities for design and process planning. We have achieved integration wherein design functions produce the required information for both activities. The integration achieved by the process-oriented approach, on the other hand, is just an automatic interface between the design and process planning activities. That is, by using the information available from the design, the interface would produce a process plan. This is undesirable from the standpoint of data consistency because it would require duplicate representation of the products, not to mention the complexity of the translation process itself.

2. Production Monitoring

The production monitoring stage of manufacturing includes activities such as quality control, scheduling, and shop floor control. Production monitoring gets its prominence from the effect that misutilization of resources and missed due dates have on the profitability of a company. Due to increasing costs and shrinking market shares, a lot of emphasis is being put on this aspect of manufacturing.

The focal point of production monitoring is the scheduling function. We have already discussed the current approaches to scheduling in Chapter II and indicated the infeasibility of producing optimal schedules which have to consider unforeseen events which may occur (such as machine breakdown). We will present an alternative approach to scheduling which uses the shop floor layout to allocate manufacturing resources to process plan components. We will then show how our data model handles the semantics of shop floor layout and conclude this section with an example to demonstrate our approach.

a. Our Approach to Scheduling and Shop Floor Layout

Our approach to scheduling is to represent the shop floor layout as an acyclic directed graph, where the nodes of the graph represent machines and the edges of the graph depict the transportation media between the machines. Each node and edge in the graph can have a job allocated to it. Figure 73 provides an example of a portion of a shop floor. The boxes labelled "pallet" depict nodes which are pallet pools for machine centers. This figure could have been more detailed, with nodes for the individual machines in the machine centers.

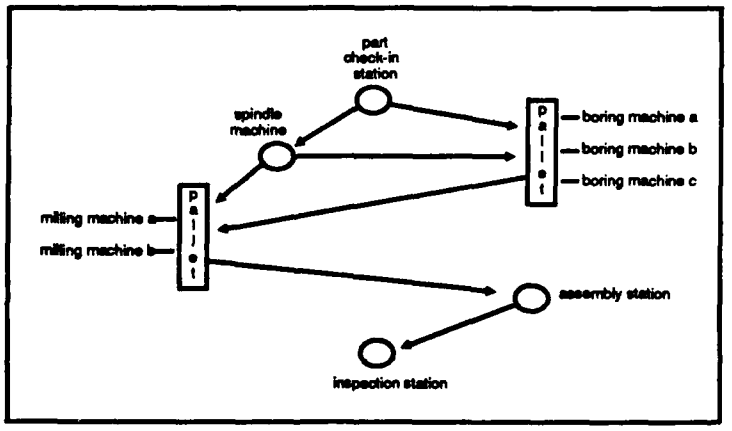


Figure 73. Shop Floor Layout

The process plans for all products currently being manufactured form a subgraph of the shop floor graph at a given point in time. The components of the process plan are divided into three categories, *working*, *ready*, and *waiting*. The working components are those which have been allocated a resource. The working components form a subgraph of the shop floor graph. The ready components are those which could begin execution if the required resources were available. The waiting components have prerequisite components and cannot be scheduled until those other components have completed processing, at which time they become ready components. Figure 74 shows a process plan, depicted as an acyclic directed graph. The components of the process plan are labelled **check-in**, **spindle**, **milling**, **boring**, and **assembly**. Initially, the check-in component will be the only ready component, the others will be waiting. Once the check-in has been completed, the spindle and milling components become ready, and can be scheduled concurrently if the proper resources are available.

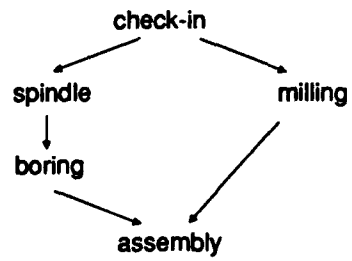


Figure 74. Sample Process Plan

The scheduling problem involves overlaying the ready components onto the shop floor graph, looking for nodes with more than one component. The presence of such a node would indicate that two or more jobs require the same resource at the same time. For these nodes, the standard priority rules could be invoked to determine which job will be assigned to them. The problem of scheduling now becomes a problem of modeling the competing process plans and manufacturing resources for each designated unit of time. This we believe is a major advantage of our approach. That is, the

scheduling problem is reduced to a simulation of the competing process plans and available resources, and uses the data made available by the integrated design and process planning functions directly.

b. Modeling the Semantics of Shop Floor Layout

We previously discussed the relationship between the design and process planning functions. The data model which we developed was used to model those functions and provided the necessary semantic facilities to combine them into a single activity. Our approach to scheduling and shop floor layout was expressed in terms of the process plans created by that activity. We will continue by showing how our data model handles the semantics of shop floor layout, the activity which drives the scheduling function and determines the overall efficiency of production.

We will demonstrate the applicability of our data model by considering the shop floor layout process. The resources available to be configured can be arranged into a hierarchy of types using the generalization/specialization abstraction concepts. The resources will be grouped into manufacturing cells to implement the flexible manufacturing system (FMS) concept discussed in Chapter II. Each cell will appear in a version hierarchy for the type of cell involved, and will use the aggregation abstraction concept to associate the various resources which are the components of that cell. New cell layouts can either be designed from scratch or developed by modifying a previously designed cell. Once again, the situation is analogous to that seen previously in the discussions on product design and process planning semantics. In fact, we now find ourselves using the system we have described to design the shop floor layout; a design which will eventually become part of the overall manufacturing system.

Individual manufacturing cells will be designed using a conceptual schema as a guide. After all of the individual cells are completed, they will be aggregated to form a layout of the shop floor. The information produced during the entire layout process can be saved and used again at two different levels. Information on individual cells can be modeled separately from the information about how those cells are aggregated to form a shop floor layout.

c. An Example

We will demonstrate our approach by use of an example. Figure 75 depicts the shop floor to be used throughout this example. Transportation resources are not shown in the figure but are available to move batches from one machine resource to another. The ovals under a particular heading represent instances of that type of machine; i.e., there are four actual milling machines on the shop floor. As each resource is used, the attributes which define its implementation change to reflect the new state of the machine. For example, the tool type attribute of a machine may vary from one job to another.

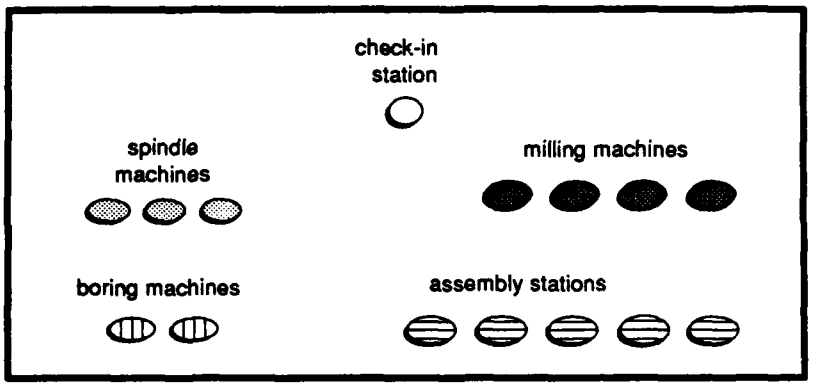


Figure 75. Shop Floor

Figure 76 shows the average processing time per workpiece required for each resource in this example. In actuality, that number could vary among machines of the same type, especially in the case where a machine is operating below its normal capacity due to need for repair. Newer machines of a given type may also be more efficient and therefore require less time to perform a particular operation. Our simulation will vary capacities among similar machines to reflect these types of conditions. Figure 77 shows the number of time units required for transportation of batches from one resource to another.

resource	time units required per piece
spindle machine	2
boring machine	1
milling machine	5
assembly station	4

Figure 76. Average Processing Time

from	to	time units required per batch
check-in	spindle machine	2
check-in	boring machine	2
check-in	milling machine	3
spindle machine	boring machine	2
spindle machine	milling machine	1
spindle machine	assembly station	2
boring machine	milling machine	2
boring machine	assembly station	1
milling machine	assembly station	2

Figure 77. Transportation Requirements

Figure 78(a) - (e) shows the five process plans to be scheduled in this example. Figure 79 provides priority and finished product quantity information for each process plan.

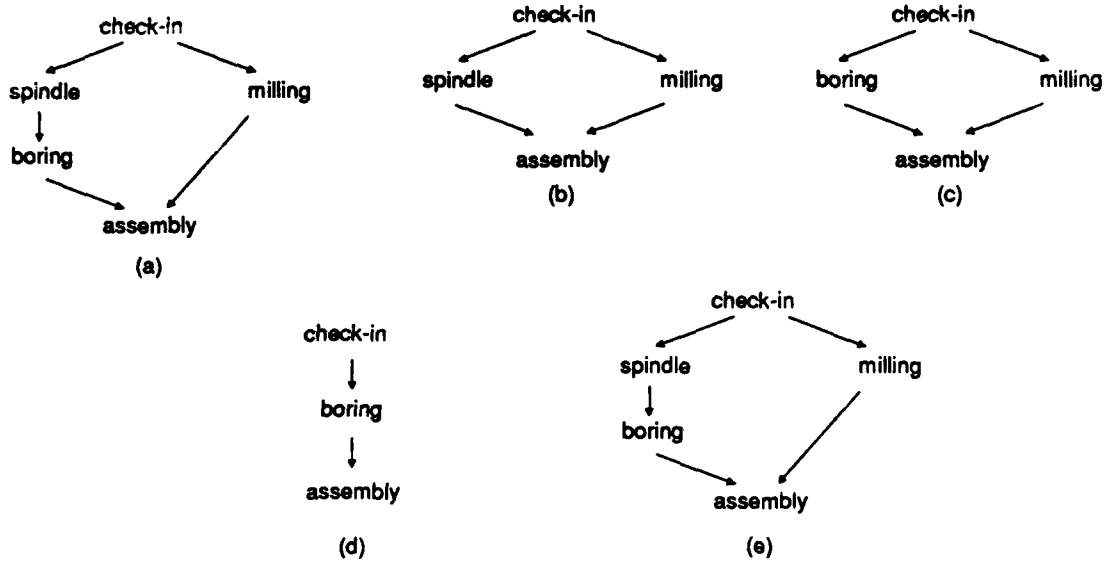


Figure 78. Example Process Plans

process plan	part quantity	priority (1 is highest)
A	10	1
B	5	5
C	12	1
D	3	4
E	15	3

Figure 79. Priority and Finished Product Information

We will assume that the necessary raw materials have been checked in and that the first task for the scheduler is to transport those materials to the appropriate resource. A simulation, written in Prolog, will show the sequence of events as the

scheduler proceeds. Besides the data from Figures 76, 77, and 79, and the process plan information shown in Figure 78, the simulation has information about each individual machine on the shop floor describing the time required for that machine to complete a given operation with a particular tool. The Prolog program for the simulation is included in the data-oriented prototype program listing attached as Appendix C.

In the first time unit, the transportation system moves materials from the check-in station to the spindle station for process plan (a), and to the boring and milling stations for process plan (c). Note that since process plans (a) and (c) have the same priority, the scheduler could have moved materials to the milling station for plan (a) instead of (c). Since it takes two time units for transportation to both the spindle and milling stations, and three time units for transportation to the boring station, all machines will be idle until time unit three. After the ninth time unit, the factory floor is in full production, with each machine resource fully utilized. After the 10th time unit, one of the boring machines becomes idle and remains idle until the 13th time unit.

After the 30th time unit, production begins winding down. The spindle and boring machines are idle, having completed their work on the example process plans. After 69 time units, all milling machines are idle, and the only work remaining involves assembly stations. The simulation continues assembly work until the completion of the 98th time unit, when all work is completed.

One advantage in using this simulation approach to scheduling is that we can change the parameters for the shop floor and rerun the simulation to determine the impact. For example, if from the first simulation run we suspected that having more assembly stations would speed up the overall operation, we could add assembly stations and verify our suspicion. The simulation would detect bottlenecks and help to alleviate them. The simulation could include allowances for machine downtime due to periodic maintenance, increase in capacity of a machine due to repair, decrease in capacity of a machine due to tool setup time, and any other foreseeable events.

Another major advantage in using this simulation approach is that the scheduling to be performed can be modeled at different levels of abstraction, which

provides increased flexibility and applicability. For example, if the details of the setup time aren't important due to infrequent changes, they can be abstracted out of the problem. An advantage to our use of Prolog as the programming language for the simulator is that expert system technology can be applied when desired.

C. SUMMARY

In this chapter we described the low-level approach to integrating manufacturing functions. We described our data-oriented approach and showed that the activities of product design and process planning could be integrated using that approach. We described the application of our approach to shop floor layout, which resulted in the integration of the production monitoring activities with the already integrated design and process planning activities, thus providing for integration across the spectrum of manufacturing functions.

VIII. EVALUATION

A. COMPARISON OF DATA MODELS

We have briefly described several data models and contrasted some of their features with our own data model (see Chapters IV, V, VI, VII). In this chapter, we will consolidate and expand the discussion of current data models to provide some means of comparison with our model. We will describe some typical scenarios from the manufacturing environment and show how four of the existing data models would fail to provide the same degree of semantic support as that offered by our model.

The nature of data models precludes any quantitative comparison or evaluation of competing models [Refs. 93, 116]. Consequently, we are limited in our ability to quantitatively compare our model with those previously defined. In addition, it is not possible to address every known data model in the discussion - there are hundreds of models in existence. The models which we selected are representative of those known models.

1. Manufacturing Activities to be Modeled

The first activity we will discuss involves the initiation of a product design from scratch. We will assume that the product to be designed is not similar to any previously designed product and therefore, none of the information about previously designed products is of use.

The second activity we will address involves the product design situation where the new product to be designed has properties and/or components which are similar to those in some previously designed product. In this case, the designer will use the information (from the previous product) about the similar features as the starting point for the new design.

The next activity we will be concerned with is complementary to the activity just mentioned. In this case, as engineer will create an intermediate starting point for future work. In this scenario, the engineer might know that the product design just completed will be similar to future design efforts. The creation of this intermediate starting point would reduce the amount of redundant design work in those future designs.

The last activity we will use involves the ability to create multiple alternatives to be used temporarily in a product design. The design engineer may decide to pursue one alternative, change his mind, and switch to some other alternative. All of the alternative information must be kept and made accessible to the designer. Once the design is completed, the information about the alternatives can be discarded or archived.

2. Support Available From Existing Models

We will begin our discussion of support available from existing models by considering the relational data model. We have already stated in Chapter IV that this model suffers from limited semantic expressiveness, a serious drawback to its use in manufacturing applications. In particular, this model does not provide a means of expressing one object as the aggregation of other objects. Figure 80 depicts a **car** as an aggregation of a **body**, **wheels**, and an **engine**. A relational schema for this example would have one relation for each box in the figure. The problem with trying to associate the three relations **body**, **wheels**, and **engine** with the relation **car** is that the model has to treat the values "body", "wheels", and "engine" in two different ways. First, they have to

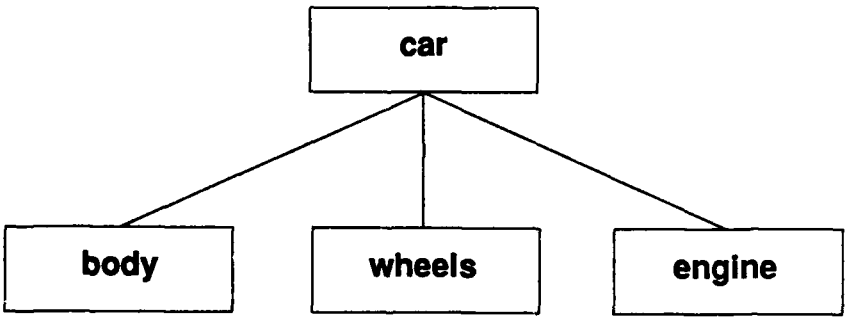


Figure 80. Sample Aggregation

be attribute values in the **car** relation to relate the car to its components. Secondly, they have to be used as relation names for the objects they represent. The standard relational model is not capable of matching the attribute values with the relation names in order to model the aggregation. This is one example of the limited semantic expressiveness of the relational model. This limitation precludes the use of this model in supporting the activities described in the previous section.

Other models have been developed which have application potential beyond that of the relational model due to their use of some of the abstraction concepts discussed previously in Chapter IV. One example is the Semantic Database Model (SDM) [Ref. 100]. SDM uses the aggregation and instantiation abstraction concepts and therefore captures more of the meaning of an application environment than is possible with the relational model [Ref. 101]. We find two major problems in trying to employ SDM in the manufacturing environment. First, the model has too many features. If a database model contains a large number of features, then it will likely be difficult to learn and to apply [Ref. 100]. Secondly, the model has no means of directly supporting the second, third, and fourth activities described in the previous section. In order to use information from a previous design, a new class would have to be defined such that the new class inherits all of the attributes, but only some of the attribute values, from the previous design. SDM has no facility for this type of inheritance. If some of the attribute values are inherited, then all of them must be. Even if the inheritance problem were overcome, SDM would have to create a new, separate class for each different set of attribute values to be inherited. The basic problem with SDM is that it has no notion of a version. Without this concept, design work always has to start from scratch and little, if any, previous information can be re-used.

The data model proposed by Katz [Ref. 46] includes support for versions, and therefore models the first, second, and third activities mentioned in the previous section. However, the fourth activity, which involves alternative designs, is not supported [Ref. 46]. In addition, only a single version of an object is maintained by the system, requiring future work to begin from a single point.

We have made repeated reference to the work by Batory and Kim [Ref. 92]. It is from this work that we get the definitions of version and instance used in our model. In addition, as we discussed in Chapter V, the two models differ in their definition of version hierarchy. The Batory and Kim definition defines version hierarchies as aggregations of other versions, allowing for flexibility in defining the implementation details for a particular product. Again, their model does not support alternative designs for the same product, a feature we feel is necessary in a data model supporting the manufacturing environment.

In the models proposed by Katz and Batory & Kim, alternative designs can be supported by the system only if the user creates and maintains the alternatives himself. It will be up to the user to remember the relationships between the alternatives and the individual identifiers for each alternative. This places an unnecessary burden on the user.

In Chapter VII we discussed the support provided by our data model for the activities mentioned in the previous section and other manufacturing activities as well. It is clear from the preceding discussion that our model captures more of the semantics of the manufacturing environment than any of these previously defined models.

B. DATA-ORIENTED VS. PROCESS-ORIENTED APPROACH

In this section we will present an example in which the data-oriented approach we have developed is more desirable than the process-oriented approach. We will use this example to illustrate the differences between the two approaches. The example we have chosen involves the design and production of a metal table. The table has three types of components, a top, some number of legs, and a connection for each leg which fastens it to the top. Since we are only interested in highlighting the differences between the process-oriented and data-oriented approaches, we will not be concerned with issues such as design integrity constraints which have to be considered in both approaches.

We have demonstrated the differences between the two approaches by implementing a prototype for each approach. In this section, we will discuss the prototype implementations, show how the process-oriented approach handles our example, and then

show how the data-oriented approach handles the same example. The discussion will be supplemented by figures which are actual screen dumps of the prototypes with colors converted to black. The actual implementations were developed using color to better distinguish information shown in the display to the user.

1. Prototype Implementations

a. General Information

The prototype implementations were written in the Turbo Prolog™ programming language. The prototypes use 640 x 350 resolution graphics with support for 16 colors. The complete program listings for both prototypes appear in Appendix C.

b. The Design Module

The two prototypes begin by allowing the user to load a conceptual schema for the product to be manufactured. The conceptual schema contains information about properties of the components of the product as well as properties of the relationships between components, where applicable. The user has the option of specifying values from scratch for the defined properties, or can load previously defined data and resume from the point at which the data was saved.

The design module screen layout contains four windows, as shown in Figure 81. The largest window, referred to as window number one in the program, shows the conceptual schema, the name of the file containing the conceptual schema data, and the name of the file containing previously defined design data, if one was loaded. Window number two appears to the right of window number one and is used as the menu window. As various design functions are performed, menu alternatives appear in the menu window. Menu alternatives are selected by positioning an arrow cursor using a mouse. Once the cursor is pointing at the desired menu alternative, pressing a mouse button will invoke that alternative. Window number three is referred to as the *data window* and appears beneath the menu window. The data window is used to display design data to the user once the **Updata Data** alternative is chosen from the top-level menu. Window number four, the *status window*, is located beneath window number one.

The status window is used to display messages to the user and to receive responses to those messages.

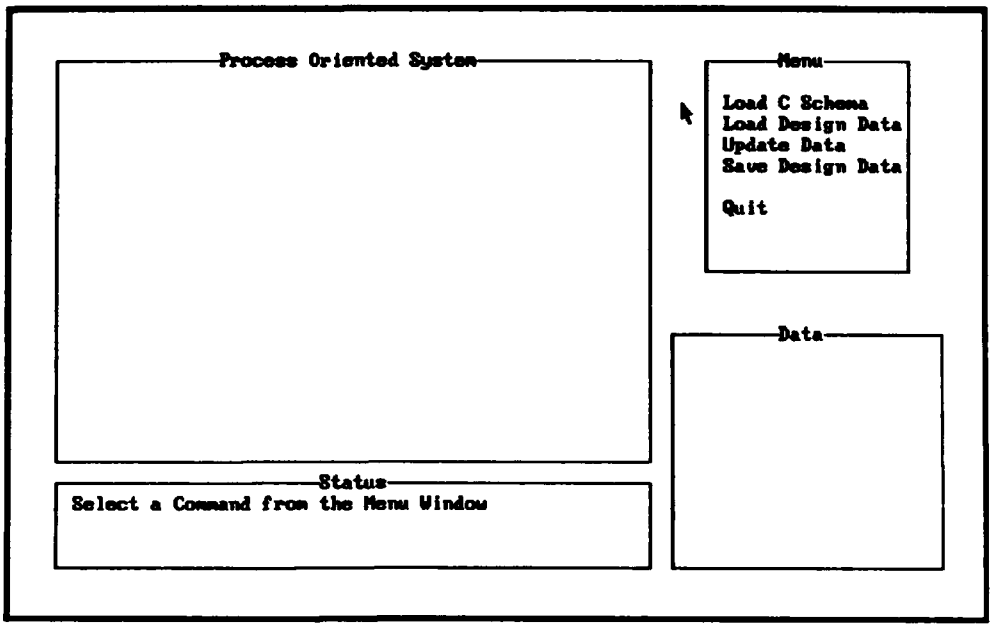


Figure 81. Initial Design Screen Layout

2. The Process-Oriented Prototype

Figure 82 shows the data interactions required by the process-oriented prototype. The product design function produces design data which is then translated into the format required by the process planning system. For the purposes of this discussion we will assume that the high-level interface described in Chapters III and VI is used to perform the translation function. The translated design data and other process planning rules are used as input to the process planning function to produce the process planning data. The difference between the process planning rules used as input and the process planning data produced as output is that the output data is specific to a given product, while the input rules are generic and assist the process planner in producing data for a specific product. The process planning data is translated using the high-level approach to produce data which can be used by the scheduling function. The scheduling rules shown

in Figure 82 refer to information about the shop floor resources which are available at the time the scheduling is performed.

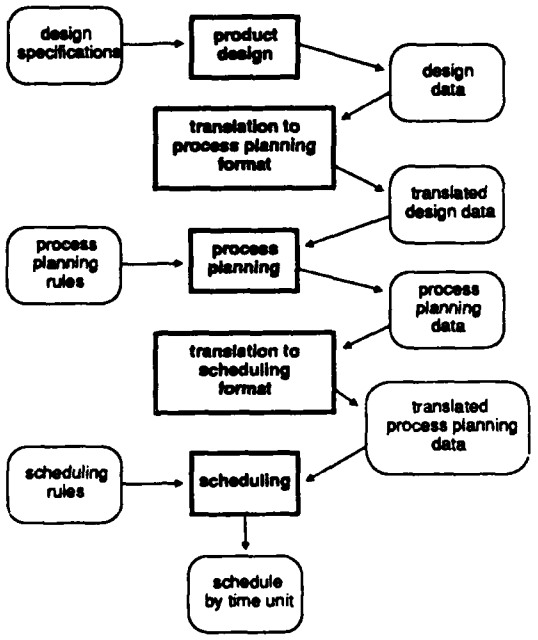


Figure 82. Process-Oriented Prototype

a. Product Design

Figure 83 shows the screen layout after the file **table.dat** is loaded using the **Load C Schema** alternative in the menu window. The conceptual schema shown in this figure represents the metal table used as our example. We will assume for simplicity sake that the designer is resuming the design using data stored previously in the file **design.dat**.

Once the design data is loaded, the designer selects the **Update Data** alternative from the menu window to continue the design of the table. Figure 84 shows the screen layout at this point. The designer will select one of the types **top**, **connect**, or **leg** by pressing the mouse button with the arrow cursor inside the appropriate conceptual schema box. In the actual implementation, the color of the text representing the selected

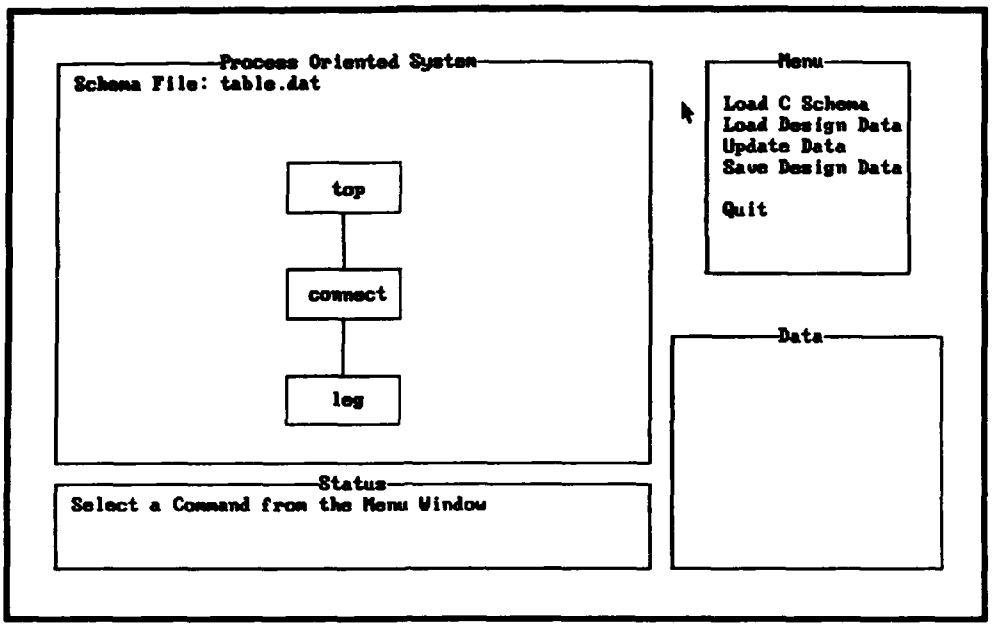


Figure 83. Conceptual Schema File Loaded

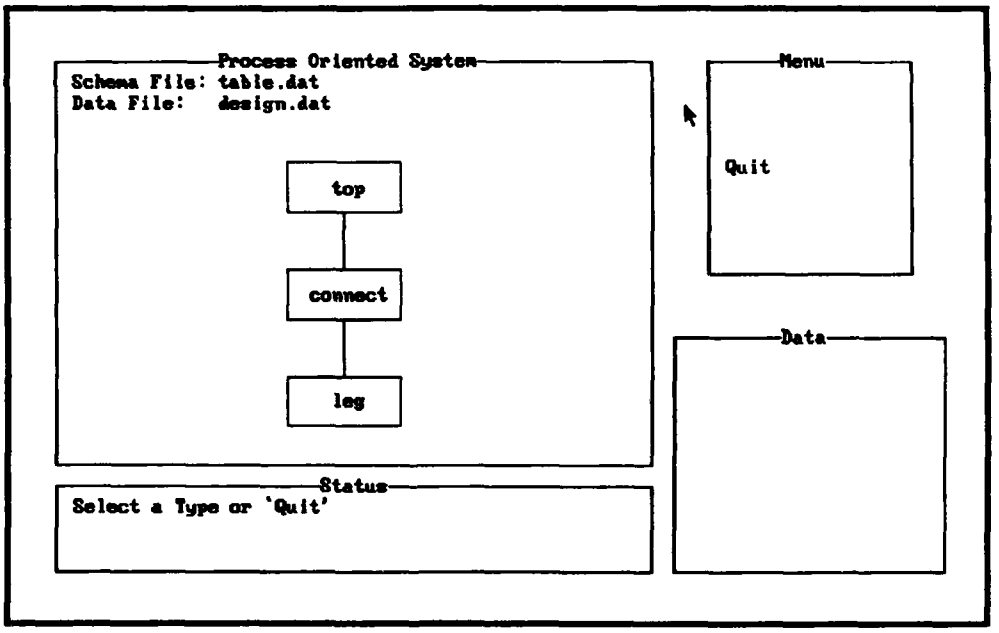


Figure 84. Second Level Menu

type is changed to distinguish it from the other types. At this point, the screen layout appears as shown in Figure 85. The user may choose to add to the existing data for the selected type, modify that data, view that data, or return to the second level menu shown in Figure 84.

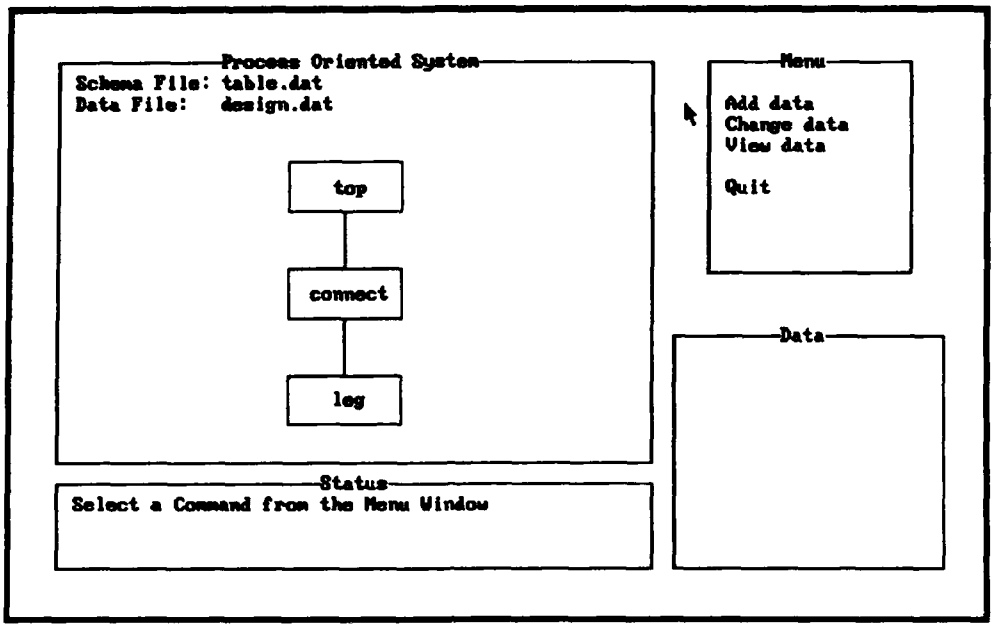


Figure 85. Third Level Menu

Figure 86 shows the screen layout after the **Add data** alternative is chosen. The data window now contains the names of the properties for the selected object. The status window is used to get user input to specify the values which the properties will take on. When the data for a single object has been entered, the screen display changes back to the display shown in Figure 85.

We will assume at this point that the designer completes the specification of the design data and saves the data in the file **design.dat**.

b. Translation of Design Data

Once the designer has finished specifying values for the properties for the objects comprising the desired product, and that data is saved, a translation process takes

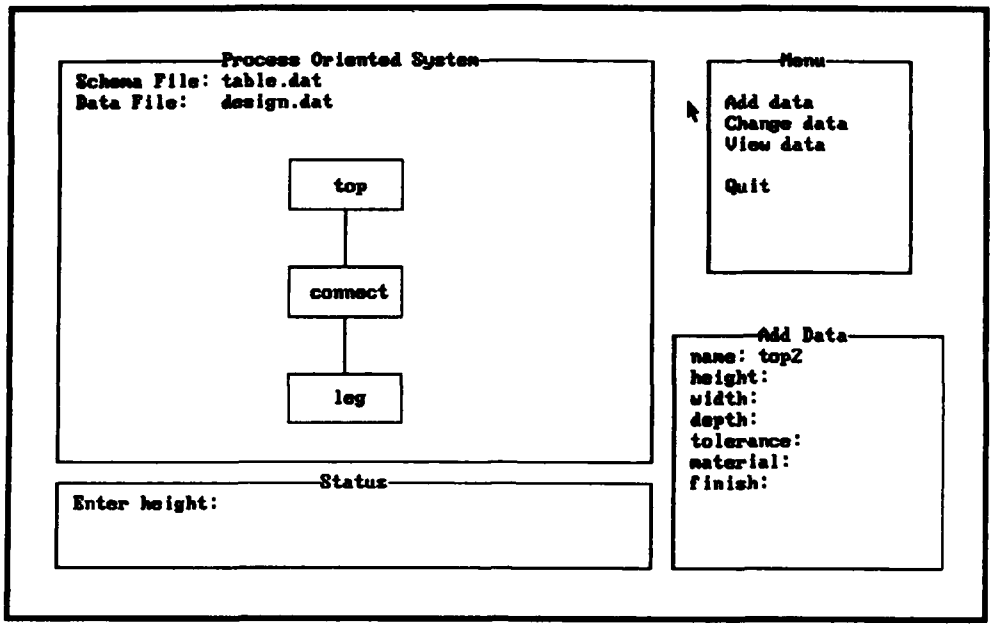


Figure 86. Adding Design Data

place which transforms the data from the format produced by the design system into the format required by the process planning system. In our prototype implementation, we assume that islands of automation exist for the design, process planning, and scheduling functions. As previously stated, we use the high-level interface approach to convert data between these islands of automation.

In the case of our example metal table, four legs have been defined which connect to the top using a welding connection for two of the legs, a bracket for the third leg, and a screw-on connection for the fourth leg. The variety of connections is used to give some diversity to the design data.

c. Process Planning

Figure 87 shows the screen display when the translated design data reaches the process planning function. Two exceptions have been detected by the process planning system which will require changes to the design data. The first exception involves the table top, where a tolerance of .1 was specified for the height, width, and

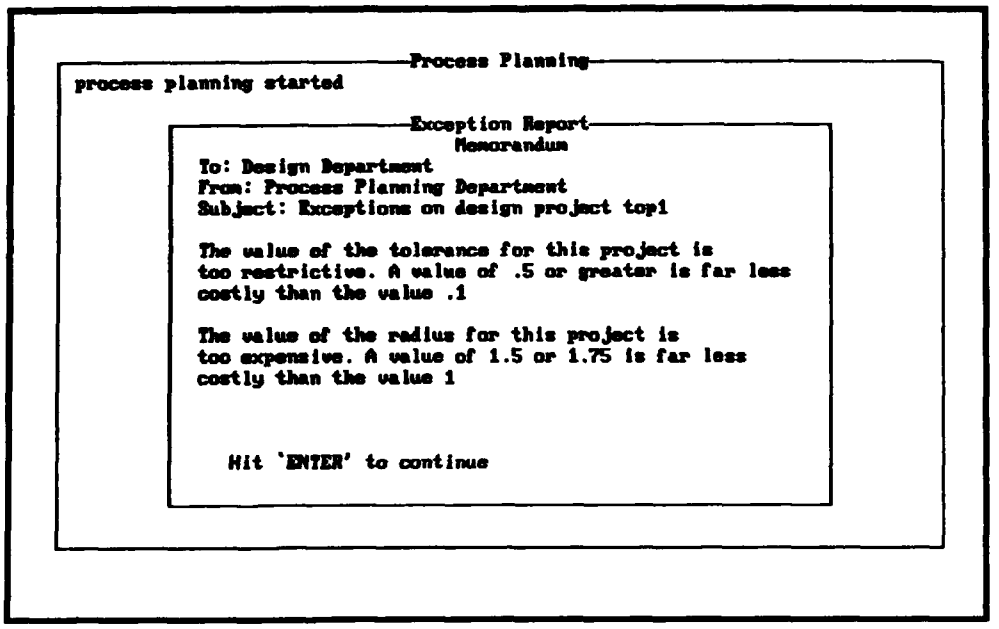


Figure 87. Process Planning Exceptions

depth dimensions. The exception notes that a value less than .5 is unacceptable due to the cost involved. The second exception involves the radius of the screw-on leg, which was specified to be one inch. The factory has a shortage of one inch radius material and a surplus of 1.5 and 1.75 inch radius material, leading to the exception.

The result of the exceptions is the automatic generation of the memorandum shown in Figure 87, which will be sent to the Design Department. The design data will have to be modified to conform to the exceptions noted in the memorandum, re-translated, and re-processed by the process planning system. Note that all of the design data will have to be re-translated and re-processed, not just the portion of the data which was modified. In our example, the design data was changed, re-translated, and no exceptions were noted by the process planning system the second time around.

d. Translation of Process Planning Data

When the product data is processed by the process planning system and no exceptions are detected, another high-level translation occurs which converts the process

planning data into the form required by the scheduling system. In the case of our example metal table, this translation occurs with no problems.

e. Scheduling

The translated process planning data is input to the scheduling system where it will compete for resources with the other products being manufactured simultaneously. Before the scheduling system can begin execution, the input data must be screened for exceptions. The screw-on leg in our example metal table has created an exception because the milling machine required to tap the screw threads is out of service. Again, a memorandum is sent to the Design Department advising them of this exception. The result of this scheduling exception, which is shown in Figure 88, is that the design data must be modified a second time, re-translated for use by the process planning system, a new process plan created, and the process planning data re-translated for input to the scheduling system. In our example, the design modifications are made, the data is re-translated and the scheduling function produces a schedule showing which resources will be allocated to the metal table and its competing products during production.

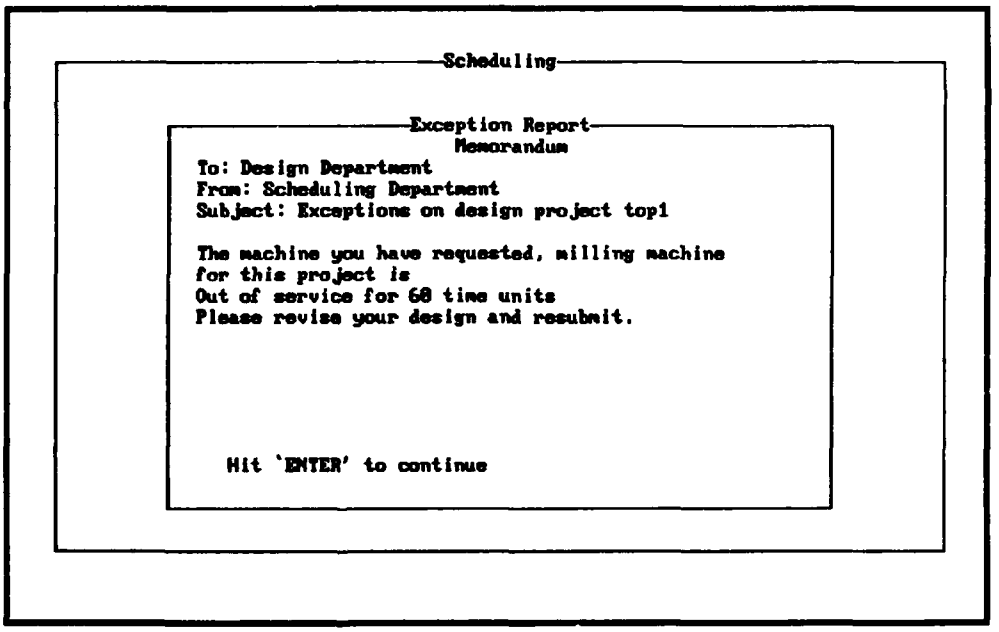


Figure 88. Scheduling Exception

3. The Data-Oriented Prototype

Figure 89 shows the data interactions required by the data-oriented prototype. The product design function takes the design specifications, process planning rules, and scheduling rules as input and produces process planning data using the values of the object properties entered by the user during the design process. The scheduling function, implemented using the simulation approach discussed in Chapter VII, uses the process planning data to produce the final schedule.

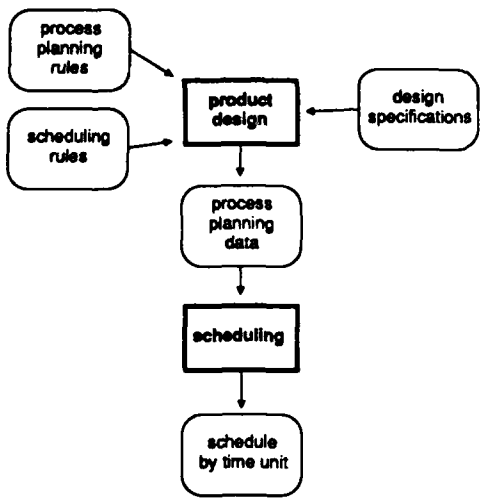


Figure 89. Data-Oriented Prototype

a. Product Design

The design function in the data-oriented approach operates similarly to its counterpart in the process-oriented approach with a few exceptions. Figure 90 shows the screen layout for data-oriented design after the conceptual schema and design data have been loaded. Up to this point, there is no difference in the operation of the two design functions.

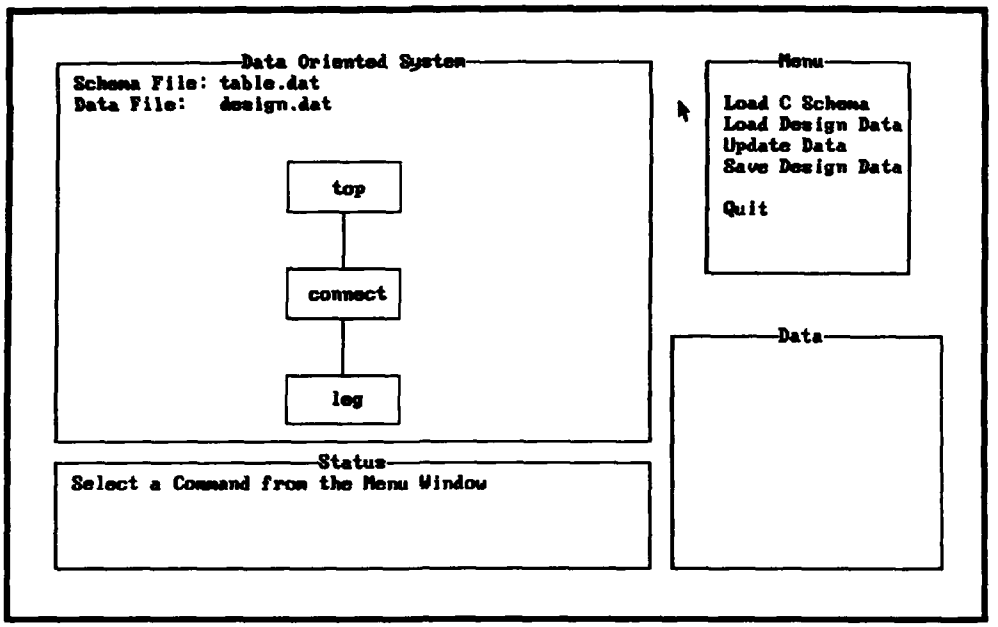


Figure 90. Data-Oriented Design

Figure 91 shows the screen layout after the **Update Data** alternative is selected from the menu window, **top** is selected, the **Change data** alternative is selected from the menu window, and the name **top1** is entered. At this point the user wants to change the **tolerance** of the dimensions to .35, so that value is entered in the status window in response to the prompt shown there. Figure 92 shows the result of entering that particular value - it is rejected. Note that when an exception occurs, the design system will not permit the user to proceed until the exception has been removed. One result of this mode of operation is that any design data which is saved has to have been free of exceptions when it was created. It is still possible that when the design data is used by the scheduling function exceptions will arise, but at least they will be minimized.

Each of the other exceptions noted in the execution of the process-oriented prototype will displayed to the designer as the values of properties are entered. This, in effect, gives the designer access to all of the pertinent information about the production of a product while it is being designed.

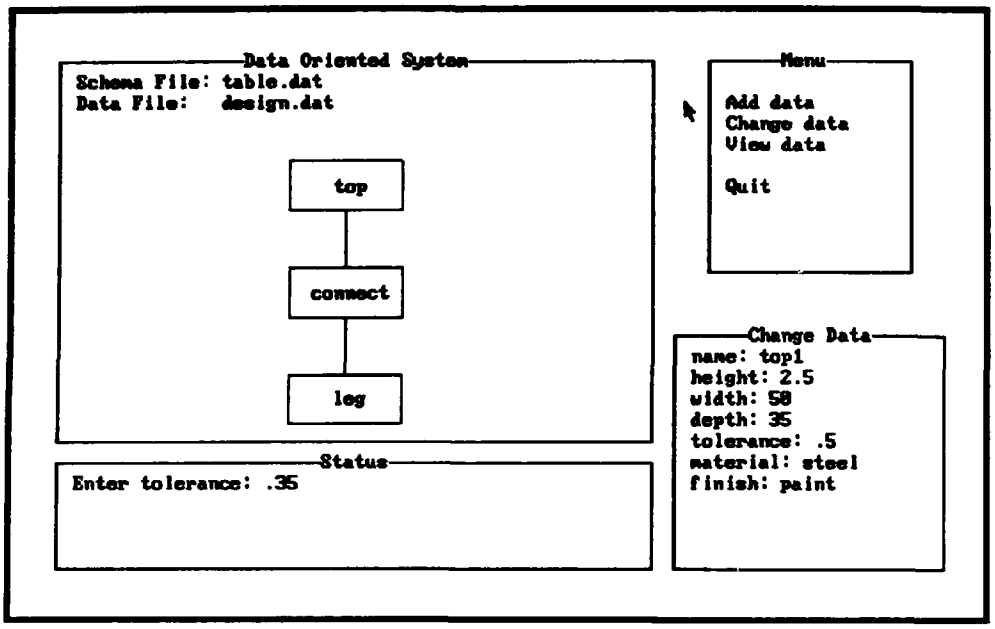


Figure 91. Specifying a Property Value

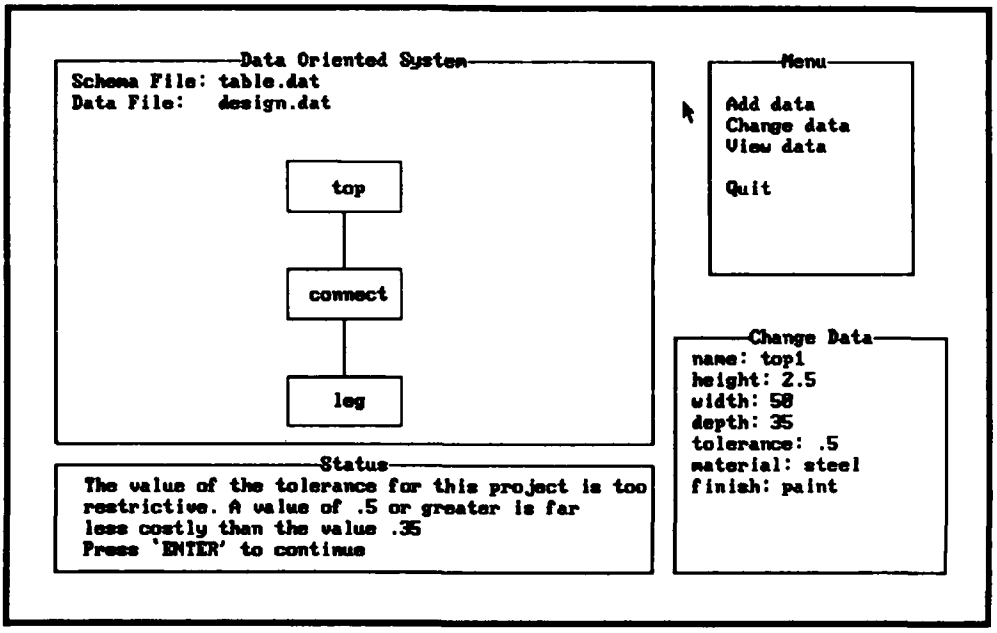


Figure 92. Process Planning Exception

b. Process Planning

The use of our low-level data-oriented approach allows the process planning function to be integrated with the design function into a single activity, as was discussed in Chapter VII. In our data-oriented prototype, this means that the process planning data required for the scheduling function is output from the design process, which we take to be the integration of design and process planning. As was shown in Figure 89, process planning still has a separate database of information used to perform process planning functions. However, this database is used by the product design function.

c. Scheduling

The data-oriented approach allows the scheduling function to use the data produced by the product design function directly. Figure 93 shows the initial scheduling display created by the scheduling simulator. The time unit is shown in the upper right corner of the screen. The **check-in** station shows one rectangle for each process plan to be scheduled. Each machine resource has one input and one output queue for each process plan to be scheduled, and the number of resources of a given type are represented by squares next to the machine name. In the original implementation, color is used to distinguish the different process plans being scheduled. Every movement of material and machine operation for a particular process plan can be monitored by watching the various queues and machine resources for the color of the desired process plan.

Figure 94 shows the screen display during the scheduling process at time unit 20. Each series of five square dots represents transportation resources being used. Again, in the original prototype, these dots are colored to correspond to the process plan utilizing the transportation resource. The same is true for the solid rectangle shown inside a machine resource square. The solid rectangle is colored so that the viewer can tell which process plan is utilizing a particular machine resource.

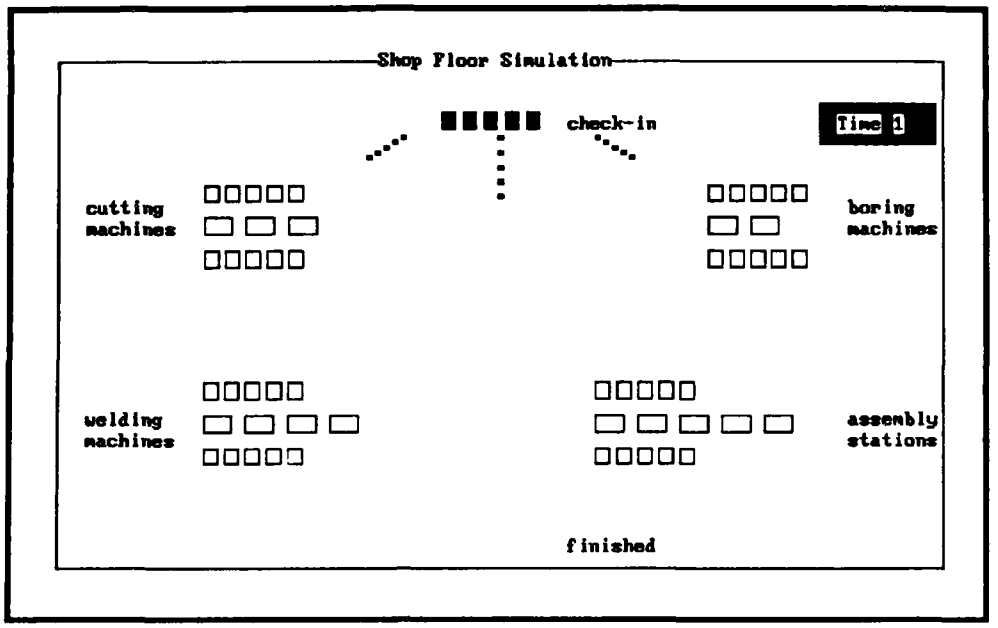


Figure 93. Initial Scheduling Display

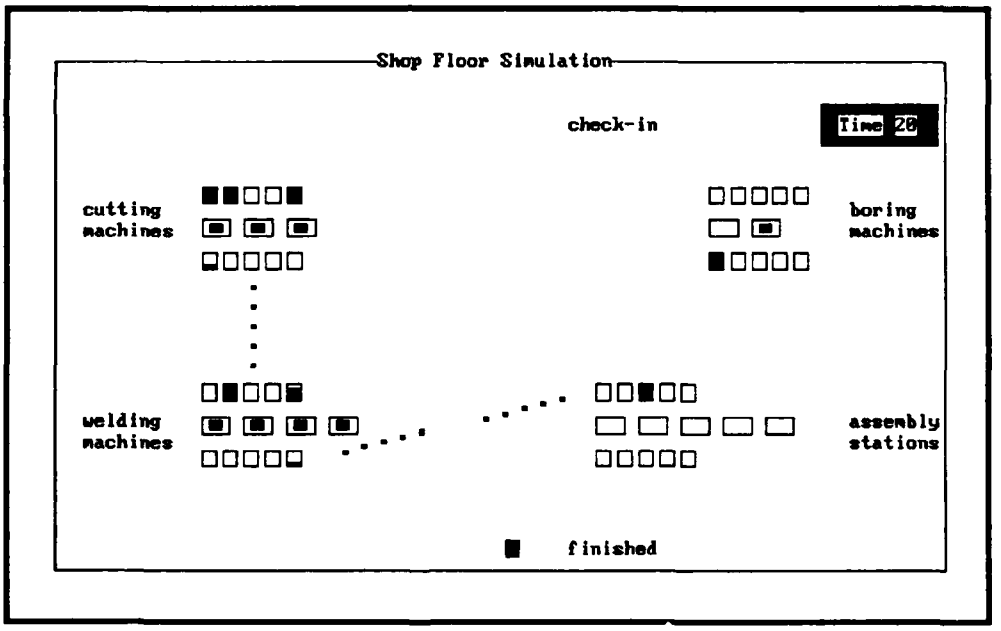


Figure 94. Scheduling Display at Time Unit 20

Figure 95 shows the scheduling screen display at the completion of time unit 42. Note that several input queues at the assembly stations and welding machines are full, but no work is being done by the resources there. This happens when all of the materials for a particular operation are not physically in the same place. For example, the welding of legs to the table tops cannot begin until all of the legs have been cut and transported to the welding machines, where the tops are waiting. At the same time, the legs which are to be attached by brackets to the table top are waiting at the assembly station for the welding to be completed. Figure 96 shows the scheduling screen display at the completion of the scheduling run.

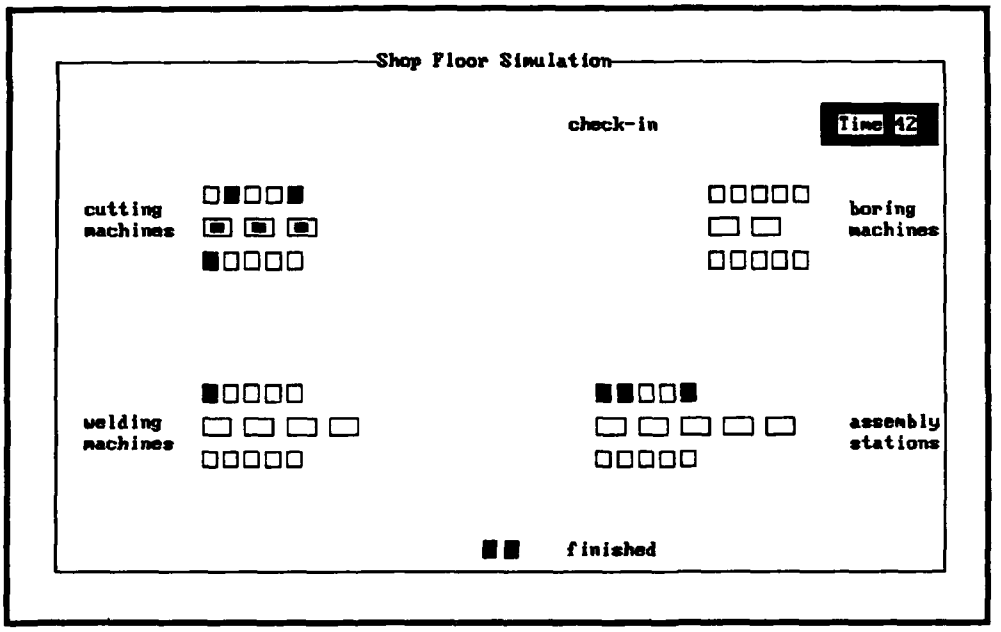


Figure 95. Scheduling Display at Time Unit 42

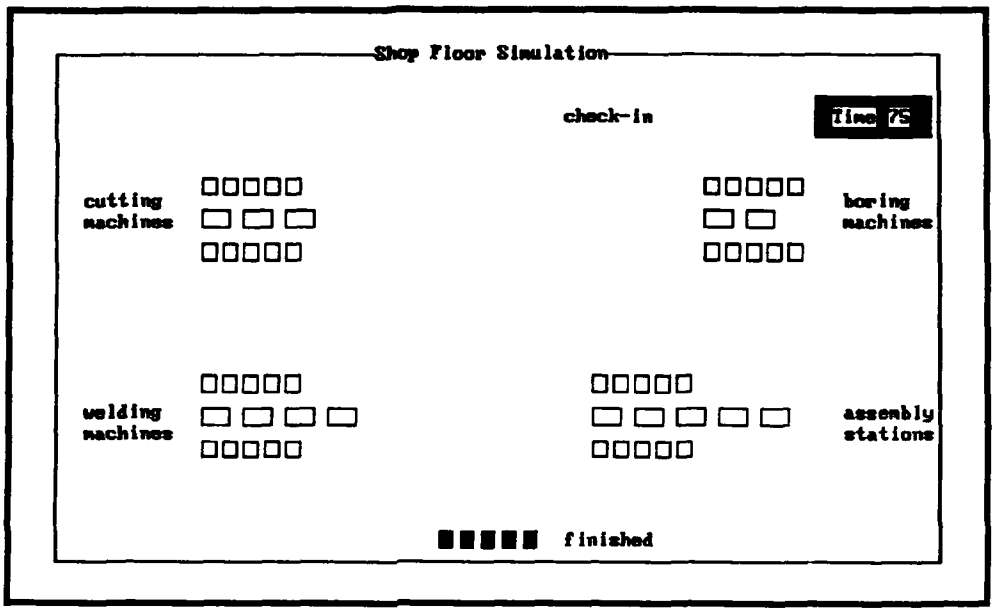


Figure 96. Display at Completion of Scheduling

4. Summary

The prototypes we have implemented point out the advantages of using the data-oriented approach. Using our example metal table and the process-oriented approach, the design data was translated three times by one translator, run through the process planning system three times, translated by the second translator twice, and processed by the scheduling system twice. The data-oriented approach avoided all of the translation and the repetitive process planning and scheduling effort by making data available where it was needed.

VIII. CONCLUSION

A. SUMMARY

It is clear that manufacturing companies need to react to changes much faster and in a more flexible way than in the past due to increasing worldwide competition, decreasing market shares, and shortage of qualified industrial and manufacturing engineers. Businesses are becoming increasingly complex due to the exponential growth rate of the technologies supporting them. How can the information flow, design, and manufacturing processes of such businesses be optimized, while maintaining marketplace presence, increasing productivity, and decreasing production costs? There is no perfect solution in terms of automation alone. Computer Integrated Manufacturing provides a short and long-term approach to a solution. The major problems to be solved are all related to integration and to providing for the possibility of further developments in technology. Those integration problems will be more easily solved when powerful computers are linked with the machinery on the shop-floor and with the factory organization as a whole.

We have described and demonstrated our data-oriented approach to the integration of manufacturing functions. Our data model, presented as a series of data abstraction concepts, clearly captures the semantics of the manufacturing environment and provides a common kernel around which those manufacturing functions can revolve. Our data-oriented perspective allowed us to conceptualize, re-organize, and simplify the product design and production process. The description of our low-level interface approach demonstrated the reduction in complexity which results from the use of our paradigm. In short, one of our major contributions is breaking the "mind set" of the traditional process-oriented approach to integration.

The application of our approach will not be easy for manufacturing companies to undertake. We expect that our perspective of the manufacturing process, if adopted, will cause some initial turmoil as steps are taken toward integration. There will always be

resistance to change and learning curves to deal with. The anticipation of impending control itself will cause anxiety and concern among employees. However, the benefits to be derived by making the change to our approach, which include better decision-making ability as a result of better control over information, should outweigh the initial investment. Employees will be more productive and the use of other manufacturing resources will be optimized because of the increased availability to relevant information afforded by the use of our data model. Again, our contribution to manufacturing will be a solution to the integration problem which makes these improvements a reality.

We have stated that the traditional database management systems lack the capability of expressing the structural and relationship aspects of the objects which exist in the manufacturing environment. Our solution was to identify the data requirements of various manufacturing functions and then create a data model to support them. Simply stated, our contribution to computer science in general and database systems in particular is the solving of a complex problem using a novel data-oriented approach.

B. EPILOG

When we first considered doing research in semantic data modeling, we didn't have any particular goal in mind other than a general goal of developing a database management system capable of supporting advanced application areas such as office automation, cartography, and CAD/CAM. We looked at the major semantic models which were previously developed and noticed that most of those models were designed to support specific application environments such as VLSI design. The abstraction concepts they included were not easily applicable and in some cases the models themselves were not easily extendible to increase their functionality. Our solution was to take parts of various models to create a new model and then supplement that model with our own abstraction concepts to increase its modeling power and therefore, its applicability.

We then considered the advanced application areas to which the model would be applied and decided on CAD/CAM. We choose house construction as the example for the application of our model since we could relate to it more readily than to other

industrial manufacturing examples. Our concept was to develop a user interface which could be used to "walk" a perspective home buyer through a model house. The user would see the features of the house just as if he were actually there. The placement of fixtures, doors windows, walls, and wiring, plumbing, and heating runs, etc., could be displayed at different levels of detail, according to the user's level of interest. The idea was to have a complete prototype of the house displayable on a high-resolution graphics screen before construction actually began. The house would have been previously designed using our proposed data model to specify and relate the various components involved. We agreed that our concept was nice and desirable, but was not a significant research problem.

During this initial stage of our research we also considered using a formal language approach to the problem of internal representation of design objects. The terminal symbols of the language would correspond to the primitive design elements of the construction environment, e.g., boards, nails, etc. The nonterminals would represent subassemblies of those primitive components and the grammar rules would specify the restrictions on the use of primitives and sub-assemblies in producing higher order complex objects. The language itself could be a context-free or modified context-sensitive language. This idea was put on the back burner since it was more implementation oriented than we wanted to deal with at the time. It is still on the back burner and will be pursued by the author as a follow-on to this thesis research.

It occurred to us that what we lacked was an overall project showing our direction, which we could chip away at, one piece at a time. We had noticed this characteristic in most of the work being presented at conferences -- they were reporting on some small aspect of research which was part of a much larger project. This led to the development of our high-level approach to integration. We laid out a diagram of the major manufacturing processes which make up CAD and CAM and decided to build a translator between the two which would produce the bill-of-materials and operations sequence information required for CAM using the data available from CAD. We decided to use a rule-based system and subsequently wrote the translator in Prolog, using an expert system

approach. We published our first conference paper shortly afterwards, describing our data model and the translator we had developed.

Between the time when we wrote this paper and presented it at the conference, we realized how impractical it would be to use the high-level approach in an actual setting. We had already conceptualized a different approach. We researched the current literature on database work in CIM and found that most attempts at integration were similar to our high-level approach and therefore in the author's opinion, are doomed in the long run. We saw from this research that a major problem was that the product life cycle itself was never changed when automation was introduced. People were viewing integration as a machine replacement process where manual work was being automated but no consideration was given to whether or not the work could have been done more efficiently some other way. Our answer was the data-oriented approach which we now call low-level integration.

We applied our data model to each of the major functions in the product life cycle, looking for the commonalty among them. It was during this process that we discovered the close relationship between product design and process planning. Our conclusion was that the two previously separate activities could be combined into one and the product life cycle was redefined.

The relevance of our work had been shown by the referee comments provided to us in the submission of conference papers and by the continued acceptance of our work in the engineering community. We applied our paradigm to the scheduling and shop floor layout aspects of manufacturing,which we considered to be our last major hurdle. The use of our approach had significantly reduced the complexity of the scheduling problem. It was true that we could not produce an optimal schedule any more than anyone else could, but what we could do was to show how better scheduling decisions could be made using the information that is inherent in that environment. It became clear that our approach had promise. Again, the relevance of our approach was shown when a reviewer of our scheduling paper decided to use it for a concurrent programming project which he had managerial responsibility for.

C. DIRECTIONS FOR FUTURE WORK

In this kind of research one question seems to surface repeatedly. *What next?* The real value in research is that it is never-ending. Every time one problem is solved, new problems unfold. This is certainly true in our case. We will answer this question in two parts. The first part will deal with implementation-oriented issues; those which will eventually lead to a working system. The second will deal with research-oriented issues.

In Chapter VII, we demonstrated how our model handles the semantics of product design. In that discussion we used figures which depict a generic user interface which would manipulate our data model. The development of such a user interface is a significant step in implementing an overall system. It would be useful in addressing some of the research-oriented issues presented below. We will have to carefully consider the question of how to develop an overall system which implements our paradigm. While we have criticized the relational model for its lack of semantic expressiveness at the conceptual level, it will probably be the most likely choice for the physical level model in such a system.

There are two major research-oriented issues to address. The first is the extension of our approach to other application environments. There are other manufacturing technologies such as FMS to which our model could be applied. Even though FMS tries to integrate manufacturing functions, it is still furthering the islands of automation problem. There is a substantial investment in the FMS technology and we feel there may be some short-term benefits to be realized in applying our model.

We feel our data model may be well-suited to other applications outside of manufacturing as well. Since our model isn't tied to any particular representation of objects, such as 3-D, it may be useful for modeling the multimedia and software engineering environments. We believe these two applications have many of the same semantics that we have seen in our research and therefore our model could be directly applicable.

Our notions of version and version hierarchy could be especially useful in supporting the software engineering environment. Software development is a key

ingredient in that environment. The program development and maintenance aspects of software development involve making changes to programs to fix errors and increase functionality. There is a requirement to maintain both the old, unmodified program and the new program which has the changes incorporated. Our version abstraction concept will model the program modification aspect while the history of modifications could be modeled by our version hierarchy concept.

The aggregation abstraction concept is also useful for modeling certain aspects of software engineering. Using a modular approach to program development, a program itself can be viewed as an aggregation of the modules which perform the input, output, and data manipulation operations. At a higher level of abstraction, software systems can be modeled as aggregations of programs and subroutines. We are convinced that our model is powerful and flexible enough to be used to support program development.

The second research-oriented issue deals with the various integration strategies which can be used to implement our paradigm. We have stated that our low-level approach is a long-term solution. In order to realize that solution, consideration will have to be given to the proper interfacing of other systems and other research results. Strategies will have to be adopted for transitioning from the islands of automation/high-level integration if our long-term solution is to be successfully implemented.

LIST OF REFERENCES

1. Mortimer, J., ed., *Integrated Manufacture*, Springer-Verlag, New York, 1985.

2. Schlesinger, R.J. and Tiersten, S., "CIM Assaults Walls Between Front Office and Shop Floor", *Hardcopy*, Dec 86, pp. 45-58.

3. Wu, C.T., "Towards Fully-Computerized Database Maintenance for Non-Traditional Applications", *Fall Joint Computer Conference*, 1987, pp. 469-474.

4. Asimow, M., *Introduction to Design*, Prentice-Hall, 1962.

5. Rembold, U. and Dillman, R., eds., *Computer-Aided Design and Manufacturing. Methods and Tools*, Springer-Verlag, Berlin, 1986.

6. Requicha, A.A.G. and Voelcker, H.B., "Solid Modeling: A Historical Summary and Contemporary Assessment", *IEEE Computer Graphics and Applications*, 2, 2, 1982, pp. 9-24.

7. McLaughlin, H.W., "Describing the Surface: Algorithms for Geometric Modeling", *Computers in Mechanical Engineering*, Nov 86, pp. 38-41.

8. Lee, Y.C. and Fu, K.S., "A CSG Based DBMS for CAD/CAM and its Supporting Query Language", *Engineering Design Applications*, 1983, pp. 123-130.

9. *Initial Graphics Exchange Specification (IGES), Version 2.0*, National Bureau of Standards, 1983.

10. "An Interface Between Geometric Modelers and Application Programs", *CAM-International Report R-80-GM-04*, 1980.

11. Wilson, P.R., et al., "Interfaces for Data Transfer Between Solid Modeling Systems", *IEEE Computer Graphics and Applications*, 5, 1, 1985, pp. 41-51.

12. Leach, L.M., *A Language Interface for Data Exchange between heterogeneous CAD/CAM Databases*, Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, NY, 1983.

13. Groover, M.P. and Zimmers, E.W., *CAD/CAM: Computer-Aided Design and Manufacturing*, Prentice-Hall, 1984.

14. Freedman, R.S. and Frail, R.P., "OPGEN: The Evolution of an Expert System for Process Planning", *The AI Magazine*, 7, 5, 1986, pp. 58-70.

15. Sheu, P. and Kashyap, R.L., "Object-based Process Planning in Automatic Manufacturing Environments", *IEEE International Conference on Robotics and Automation*, 1987, pp. 435-440.

16. Conaway, J., "What's in a Name: Plain Talk About CIM", *Computers in Mechanical Engineering*, Nov 85, pp. 23-31.

17. Chang, T-C. and Wysk, R.A., "Interfacing CAD/Automated Process Planning", *AIIE Transactions*, Sep 81.

18. Hegland, D.E., "Out in Front with CADCAM at Lockheed - Georgia", *Production Engineering*, Nov 81.

19. Wright, P.K. and Englert, P.J., "Sharpening the Senses of Industrial Robots", *Mechanical Engineering*, May 86, pp. 58-63.

20. Ullman, J.D., "NP-Complete Scheduling Problems", *Journal of Computers and System Science*, 10, 1975, pp. 384-394.

21. Fox, B.R. and Kempf, K.G., "Reasoning About Opportunistic Schedules", *IEEE International Conference on Robotics and Automation*, 1987, pp. 1876-1882.

22. Gallimore, J., "How to Make MRP Really Work", *The Production Engineer*, Mar 84, pg. 22.

23. Leahy, J.A., "Management Issues in JIT and OPT Implementations", *Annual International Industrial Engineering Conference*, 1985, pp. 82-87.

24. Leahy, J.A., "Integrating Just In Time and Optimized Production Technology", *Fall Industrial Engineering Conference*, 1986, pp. 315-318.

25. Tompkins, J.A. and Cramer, M.A., "Just In Time: The Real Story", *CIM Technology*, Aug 87, pp. CT31-CT32.

26. DeVries, M.F., et al., Group Technology, Publication MDC 76-601, Machinability Data Center, Cincinnati, Ohio, 1976.

27. Greene, T.J., "Is Cellular Manufacturing Right for You?", *Annual International Industrial Engineering Conference*, 1985, pp. 181-190.

28. Fenves, S.J. and Rasdorf, W.J., "Treatment of Engineering Design Constraints in a Relational Data Base", *Engineering With Computers*, Springer-Verlag, 1, 1985, pp. 27-37.

29. Rasdorf, W.J., "Extending DBMS's for Engineering Applications", *Computers in Mechanical Engineering*, Mar 87, pp. 62-69.

30. Date, C.J., *An Introduction to Data Base Systems*, Addison-Wesley, Vol 1, fourth ed., 1986.

31. Rasdorf, W.J., "Perspectives on Knowledge in Engineering Design", *International Computers in Engineering Conference*, ASME, Vol 2, 1985, pp. 249-253.

32. Rasdorf, W.J. and Wang, T.E., "CDIS: An Engineering Constraint Definition and Integrity Enforcement System for Relational Data Bases", *International Computers in Engineering Conference*, ASME, Vol 2, 1986, pp. 273-280.

33. Stonebraker, M., Rubenstein, B., and Guttman, A., "Application of Abstract Data Types and Abstract Indices to CAD Data Bases", *Engineering Design Applications*, 1983, pp. 107-113.

34. Traughber, T.J., "Software Design Considerations for Computer Aided Process Planning (CAPP)", *Texas Instruments Technical Report 86082*.

35. Luby, S.C., Dixon, J.R., and Simmons, M.K., "Creating and Using a Features Data Base", *Computers in Mechanical Engineering*, Nov 86, pp. 25-33.

36. Dube, R.P. and Smith, M.R., "Managing Geometric Information with a Database Management System", *IEEE Computer Graphics and Applications*, Oct 83, pp. 57-62.

37. Lorie, R. and Plouffe, W., "Complex Objects and Their Use in Design Transactions", *Engineering Design Applications*, 1983, pp. 115-121.

38. Lorie, R., et al., "Supporting Complex Objects in a Relational System for Engineering Databases", in *Query Processing in Database Systems*, Springer-Verlag, New York, pp. 145-155.

39. Taraman, S.R., "A Relational Data Model for a Manufacturing Database", *Annual International Industrial Engineering Conference*, 1985, pp. 195-203.

40. Wedekind, H. and Zoerntlein, G., "Conceptual Basis for Database Applications in Flexible Manufacturing Systems (FMS)", *IEEE CH 2413-3*, 1987, pp. 551-557.

41. Tulkoff, J., "Process Planning in the New Information Age", *CIM Technology*, Aug 87, pp. CT19-CT22.

42. Brodie, M.L. and Mylopoulos, J., *On Knowledge Base Management Systems*, Springer-Verlag, New York, 1986.

43. Barkocy, B.E. and Zdeblick, W.J., "A Knowledge-Based System for Machine Operation Planning", *Proceedings of AUTOFACT 6*, 1984, pp. 2.11 - 2.25.

44. Phillips, R.H. and Mouleeswaran, C.B., "A Knowledge-Based Approach to Generative Process Planning", *Proceedings of AUTOFACT 7*, 1985, pp. 10.1-10.15.

45. Ebeid, S.J., "Computerized Machinability Data Base Systems as Applied to Non-Conventional Machining Processes", *Association for the Advancement of Modelling and Simulation Techniques in Enterprises, Review*, 4, 1, 1986, pp. 7-17.

46. Katz, R.H., "A Database Approach for Managing VLSI Design Data", *19th Design Automation Conference*, 1982, pp. 274-282.

47. Stonebraker, M. and Guttman, A., "Using a Relational Database Management System for CAD Data", *IEEE Database Engineering Bulletin*, 1982.

48. McLeod, D., Narayanaswamy, K., and Bapa Rao, K.V., "An Approach to Information Management for CAD/VLSI Applications", *Engineering Design Applications*, 1983, pp. 39-50.

49. Kent, W., "Limitations of Record-Based Information Models", *ACM Transactions on Database Systems*, 4, 1, Mar 79, pp. 107-131.

50. Afsarmanesh, H., et al., "An Extensible Object-Oriented Approach to Databases for VLSI/CAD", *11th International Conference on Very Large Data Bases*, 1985, pp. 13-24.

51. Jones, J.E. and Turpin, W., "Developing An Expert System for Engineering", *Computers in Mechanical Engineering*, Nov 86, pp. 10-16.

52. Su, S.Y.W., et al., "The Architecture and Prototype Implementation of an Integrated Manufacturing Database Administration System, *IEEE Computer Society COMPCON*, 1986, pp. 287-296.

53. McLean, C.R. and Brown, P.F., "Process Planning in the AMRF", *CIM Technology*, Aug 87, pp. CT23-CT26.

54. Buchmann, A.P. and deCelis, C.P., "An Architecture and Data Model for CAD Databases", *11th International Conference on Very Large Databases*, 1985, pp. 105-114.

55. Vrba, J.A., "CAM for the 80's - Distributed Systems Using Local Area Networks", *AUTOFACT 6*, 1984, pp. 10.14-10.27.

56. Elgabry, A.K., "Communicating Product Definition and Support Data in A CAE/CAD/CAM Environment", *AUTOFACT 7*, 1985, pp. 9.11-9.26.

57. Baer, T., "MAPping the Factory", *Mechanical Engineering*, Jan 86, pp. 70-73.

58. Mulvey, P. and Sheftic, R., "Information Flow on the Factory Floor: A Network for Automation", *Computers in Mechanical Engineering*, Jul 86, pp. 15-19.

59. Shah, M.J. and Brecher, V.H., "Distributed Computer Control in Manufacturing", *Computers in Mechanical Engineering*, Nov 85, pp. 50-56.

60. Williams, T.J., "Recent Developments in the Application of Plant-Wide Computer Control", *Computers in Industry*, Apr 87, pp. 233-254.

61. Fong, J.T., "Integration of Analysis and Data Bases for Engineering Decision Making", *Computers in Mechanical Engineering*, Jul 86, pp. 42-55.

62. Brown, D.C. and Posco, P., "Looking for Trouble: Expert Browsing In Manufacturing Data Bases", *Computers in Mechanical Engineering*, Nov 86, pp. 19-23.

63. Davis, R.P., et al., "Manufacturing Systems Planning - The Key to Production Control", *International Industrial Engineering Spring Annual Conference*, 1979, pp. 295-302.

64. Barish, M.M., "Computerized Systems in the Scheme of Things", *International Industrial Engineering Fall Conference*, 1979.

65. Allen, M.A., *Keynote Address*, WESTEC '84, Mar 84.

66. Greene, T.J., "CIM Goes Beyond Production and Inventory Control", *Production and Inventory Control Division Newsletter*, Institute of Industrial Engineering, vol XIX, no 1, 1984.

67. Clancy, J., quoted in *Industry Week*, 224, 5, 1985, pg. 52.

68. Conaway, J., quoted in *Industry Week*, 224, 5, 1985, pg. 49.

69. Saxe, K., "MRP-II Into CIM: The Interface Phase", *AUTOFACT 7*, 1985, pp. 3.1-3.6

70. Marshal, J. and Van Dyne, D., "Integrating CAE, CAD and CASE", *Digital Design*, Jun 86, pp. 40-46.

71. Okogbaa, O.G., et al., "Scheduling Rules for Just-In-Time Policy in a Computer Integrated Manufacturing System", *International Industrial Engineering Conference*, 1985, pp. 137-144.

72. "CADCAM - Clarifying the Connection", Staff Report, *Mechanical Engineering*, Mar 87, pp. 45-47.

73. Beeby, W., "The Future of Integrated CAD/CAM Systems: The Boeing Perspective", *IEEE Computer Graphics and Applications*, Jan 82, pp. 51-56.

74. "Computer Integrated Manufacturing: The Focus of Manufacturing Control", *Assembly Engineering*, May 83, 26,5, pp. 12-16.

75. Gondert, S.J., "Understanding the Impact of Computer-integrated manufacturing", *Manufacturing Engineering*, 93, 3, Sep 84, pp. 67-69.

76. Schroeder, C., "CADMAC, Next Step on the Path to CIM", *AUTOFACT 7*, 1985, pp. 9.41-9.63.

77. Pinte, J.M., "A Computer-Integrated Manufacturing System for the Metalworking Industry", *AUTOFACT 7*, pp. 3.35-3.53.

78. Kusiak, A., "Design and Management of Computer Integrated Manufacturing Systems", *Fall Industrial Engineering Conference*, 1986, pp. 353-361.

79. Tsichritzis, D.C. and Lochovsky, F.H., *Data Models*, Prentice-Hall, 1982.

80. Langefors, B., "Information systems theory", *Information Systems*, 2, 1977, pp. 207-219.

81. Bubenko, J.A., "The Temporal Dimension in Information Processing, Architecture and Models in Database Management, G.M. Nijssen, Ed., North-Holland, 1977, pp. 93-118.

82. Abrial, J.R., "Data Semantics", in *Data Base Management*, North-Holland, 1974, pp. 1-59.

83. Yao, S.B., ed, *Principles of Database Design*, vol 1, Prentice-Hall, 1985.

84. *CODASYL Data Base Task Group Report*, Conference on Data Systems Languages, Association for Computing Machinery, 1971.

85. Maier, D., *The Theory of Relational Databases*, Computer Science Press, 1983.

86. Su, S.Y.W., "Modeling Integrated Manufacturing Data with SAM*", *IEEE Computer*, Jan 86, pp. 34-49.

87. Biller, H. and Neuhold, E.J., "Semantics of Databases: The Semantics of Data Models", *Information Systems*, 3, 1978, pp. 11-30.

88. Smith, J.M. and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, 2, 2, Jun 77, pp. 105-133.

89. SOWA, J.F., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, 1984.

90. Presman, R.S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 1982.

91. Mylopoulos, J., Bernstein, P.A., and Wong, H.K.T., "A Language Facility for Designing Database-Intensive Applications", *ACM Transactions on Database Systems*, 5, 2, 1980, pp. 185-207.

92. Batory, D.S. and Kim, W., "Modeling Concepts for VLSI CAD Objects", *ACM Transactions on Database Systems*, 10, 3, Sep 85, pp. 322-346.

93. Brodie, M.L., "Association: A Database Abstraction for Semantic Modelling", *2nd International Entity-Relationship Conference*, 1981, pp. 577-602.

94. Madison, D.E. and Wu, C.T., "An Expert System Interface and Data Requirements for the Integrated Product Design and Manufacturing Process", *IEEE International Conference on Data Engineering*, 1987, pp. 610-618.

95. Chen, P.P.S., "The Entity-Relationship Model - Toward a Unified View of Data", *ACM Transactions on Database Systems*, 1, 1, Mar 76, pp. 9-36.

96. Shipman, D.W., "The Functional Model and the Data Language DAPLEX", *ACM Transactions on Database Systems*, 6, 1, Mar 81, pp. 140-173.

97. Buneman, P. and Frankel, R.E., "FQL - A Functional Query Language", *ACM SIGMOD International Conference on the Management of Data*, 1979.

98. Housel, B.C., Waddle, V., and Yao, S.B., "The Functional Dependency Model for Logical Database Design", *Fifth International Conference on Very Large Data Bases*, 1979, pp. 194-208.

99. Brodie, M.L., "On Modelling Behavioural Semantics of Data", *7th International Conference on Very Large Data Bases*, Sep 81, pp. 32-42.

100. Hammer, M. and McLeod, D., "Database Description with SDM: A Semantic Database Model", *ACM Transactions on Database Systems*, 6, 3, Sep 81, pp. 351-386.

101. Borgida, A., "Features of Languages for the Development of Information Systems at the Conceptual Level", *Technical Report LCSR-TR-52*, Laboratory for Computer Science Research, Rutgers University, Dec 83.

102. Su, S.Y.W. and Lo, D.H., "A Semantic Association Model for Conceptual Database Design", *International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, Dec 79.

103. Codd, E.F., "Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems*, 4, 4, Dec 79, pp. 397-434.

104. Goldberg, A. and Robson, D., *Smalltalk-80: The Language and Its Implementation*, Addison-Wesley, 1983.

105. Davis, H.F. and Snider, A.D., *Introduction to Vector Analysis*, Allyn and Bacon, Inc., 1975, pp. 1-11.

106. Schek, H.J. and Pistor, P., "Data Structures for an Integrated Data Base Management and Information Retrieval System", *International Conference on Very Large Data Bases*, 1982, pp. 197-207.

107. Hayes-Roth, F., Lenat, D.B., and Waterman, D.A., *Building Expert Systems*, Addison-Wesley, 1983, pp. 219-235.

108. Winston, P.H., *Artificial Intelligence*, Addison-Wesley, 1984.

109. Batory, D.S. and Buchmann, A.P., "Molecular Objects, Abstract Data Types, and Data Models - A Framework", *International Conference on Very Large Data Bases*, 1984, pp. 172-184.

110. Guttman, A. and Stonebraker, M., "Using a Relational Database Management System for Computer Aided Design Data", *IEEE Database Engineering*, 5, 2, 1982.

111. Haskin, R. and Lorie, R., "On Extending the Function of a Relational Database System", ACM SIGMOD Conference on the Management of Data, 1982, pp. 207-212.

112. Madison, D.E. and Wu, C.T., "The 'Integration' in Computer Integrated Manufacturing", *International Conference on Engineering Design*, 1987.

113. Madison, D.E., Wilbur, T.G., and Wu, C.T., "Data-Driven CIM", *Computers in Mechanical Engineering*, 6, 5, 1988.

114. Madison, D.E. and Wu, C.T., "A Database Approach to Computer Integrated Manufacturing: Process Planning Using Group Technology", *International Conference on CAD/CAM, Robotics, and Factories of the Future*, 1988.

115. Madison, D.E. and Wu, C.T., "A Database Approach to Computer Integrated Manufacturing: Scheduling and Shop Floor Layout", *Naval Postgraduate School Technical Report NPS52-87-047*, submitted for publication.

116. Brodie, M. L., Mylopoulos, J., and Schmidt, J. W., editors, *On Conceptual Modeling*, Springer-Verlag, New York, 1984.

APPENDIX A - TRANSLATOR PROGRAM

A. MAIN PROGRAM

```
start :-
   not(begin_stds_check),
   not(begin_operations),
   not(set_neg_area),
   not(raw_materials_needed),
   not(materials_report),
   not(report_subst).

begin_stds_check :- kind_of(Extens,Intens),
   write('check for'),write(Intens),write(' '),write(Extens),nl,nl,
   check(Extens,Intens),fail.

check(Extens,Intens) :-
   property(Extens,material_type,Material),
   material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
   comment_for(Intens,Spec_Mat,Class),
   comment(Class,Comment),
   write('    '),write(Comment),nl,nl.

check(Extens,Intens) :-
   property(Extens,material_type,Material),
   material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
   check_for(Intens,Spec_Mat,Class),
   member(Material,Class),
   write('    '),write(Intens),write(' '),write(Extens),
   write(' meets requirements; allowed substitutes are:'),nl,nl,
   member(Other_Mat,Class),
   not(Other_Mat = Material),
   write('    - '),write(Other_Mat),nl,nl,
   assertz(substitute(Extens,Other_Mat)).
```

```
check(Extens,Intens) :-
   property(Extens,material_type,Material),
   material(Material,Spec_Mat,_,_,_,_,_,_,_,_,_,_,_),
   check_for(Intens,Spec_Mat,Class),
   not(member(Material,Class)),
   write('    '),write(Intens),write(' '),write(Extens),
   write('does not meet requirements; allowed substitutes are:'),nl,nl,
   member(Other_Mat,Class),
   write('    - '),write(Other_Mat),nl,nl,
   assertz(substitute(Extens,Other_Mat)).

check(Extens,door) :-
   dimension(Extens,Dimension,Z,Units),
   minimum(door,Extens,Dimension,X,Unitx),
   maximum(door,Extens,Dimension,Y,Unity),
   convert(X,Unitx,Min,Units),
   convert(Y,Unity,Max,Units),
   check_standards(door,Extens,Dimension,Z,Min,Max).

check(Extens,pane) :-
   property(Extens,quality,Value),
   minimum(pane,Extens,quality,Min),
   check_standards(pane,Extens,Value,Min).

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
   not(Min > Value), not(Value > Max),
   write('    '),write(Intens),write(' '),write(Extens),
   write(' passed - '),write(Dimension),nl,nl,!.

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
   Min > Value,
   write('    '),write(Intens),(' '),write(Extens),
   write(' failed minimum - '),
   write(Dimension),nl,nl,!.

check_standards(Intens,Extens,Dimension,Value,Min,Max) :-
   Value > Max,
   write('    '),write(Intens),write(' '),write(Extens),
   write(' failed maximum - '),
   write(Dimension),nl,nl,!.
```

```
check_standards(pane,Extens,Value,Min) :-
   not(Min > Value),
   write('    '),write('pane '),write(Extens),
   write(' passed quality check'),nl,nl,!.

check_standards(pane,Extens,Value,Min) :-
   Min > Value,
   part_of(Extens,Window),
   kind_of(Window,window),
   write('    '),write('pane '),write(Extens),
   write(' failed minimum quality check'),nl,
   write('      - part of '),write(Window),nl,nl,!.

begin_operations :-
   kind_of(H,house),
   not(do_assembly(H)),
   not(operations_report(H)),
   fail.

do_assembly(H) :- assemble(H,house),fail.

operations_report(H) :-
   nl,nl,
   write('*********************************************'),nl,
   write('*                                           *'),nl,
   write(' Production Sequence Report for '),
   write(H),nl,nl,
   print_style(H),
   write('*                                           *'),nl,
   write('*********************************************'),nl,nl,!,
   operation(Extens,Function,Attribute1,Attribute2),
   print_operation(Extens,Function,Attribute1,Attribute2),
   fail.
```

```
print_operation(comment,Comment,_,_) :-
   nl,
   write('*********************************************'),nl,
   write('*                                           *'),nl,
   write(' comment : '),
   write(Comment),
   nl,
   write('*                                           *'),nl,
   write('*********************************************'),nl,nl,!.

print_operation(Extens,Attribute1,Attribute2,Attribute3) :-
   write(Extens),
   name(Extens,L1),
   length(L1,N1),
   tab(15 - N1),
   write(Attribute1),
   get_name_len(Attribute1,N2),
   tab(15 - N2),
   write(Attribute2),
   get_name_len(Attribute2,N3),
   tab(17 - N3),
   write(Attribute3),nl,!.

get_name_len(Name,Len) :-
   number(Name),
   not(integer(Name)),
   name(Name,L1),
   length(L1,N1),
   Len is (N1 - 4),!.

get_name_len(Name,Len) :-
   name(Name,L1),
   length(L1,Len),!.

print_style(H) :-
   property(H,subtype,Hstyle),
   write('- house style is '),
   write(Hstyle),nl,
   write('   and consists of '),
   contains(H,L),
   write(L),nl,!.
```

```
print_style(H).

/* routines to calculate surface area of faces taken up by */
/* doors, windows, openings, and connections              */
set_neg_area :-
   kind_of(Extens,face),
   set_neg_area2(Extens,[],0,feet),fail.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,window),
   dimension(Extens,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Ht * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,door),
   dimension(Extens,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Ht * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,connection),
   geometry(Extens,rectangle),
   dimension(Extens,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Ht * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.
```

```
set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,connection),
   geometry(Extens,square),
   dimension(Extens,height,Ht,Htunits),
   convert(Ht,Htunits,New_Ht,Units),
   New_Area is (Area + (New_Ht * New_Ht)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,connection),
   geometry(Extens,square),
   dimension(Extens,width,Wd,Wdunits),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Wd * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,connection),
   geometry(Extens,circle),
   dimension(Extens,radius,Rd,Rdunits),
   convert(Rd,Rdunits,New_Rd,Units),
   Pi is 3.14159,
   New_Area is (Area + (Pi * New_Rd * New_Rd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,opening),
   geometry(Extens,rectangle),
   dimension(Extens,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Ht * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.
```

```
set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,opening),
   geometry(Extens,square),
   dimension(Extens,height,Ht,Htunits),
   convert(Ht,Htunits,New_Ht,Units),
   New_Area is (Area + (New_Ht * New_Ht)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,opening),
   geometry(Extens,square),
   dimension(Extens,width,Wd,Wdunits),
   convert(Wd,Wdunits,New_Wd,Units),
   New_Area is (Area + (New_Wd * New_Wd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   face(Extens,Face),
   not(member(Extens,L)),
   kind_of(Extens,opening),
   geometry(Extens,circle),
   dimension(Extens,radius,Rd,Rdunits),
   convert(Rd,Rdunits,New_Rd,Units),
   Pi is 3.14159,
   New_Area is (Area + (Pi * New_Rd * New_Rd)),
   set_neg_area2(Face,[Extens|L],New_Area,Units),!.

set_neg_area2(Face,L,Area,Units) :-
   assertz(get_neg_area(Face,Area,Units)).

get_area(Extens,Area,Units) :-
   part_of(Extens,Face),
   dimension(Extens,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   get_neg_area(Face,Neg_Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Area is ((New_Ht * New_Wd) - Neg_Area),!.
```

```
get_area(Extens,Area,Units) :-
   part_of(Extens,Face),
   dimension(Face,height,Ht,Htunits),
   dimension(Face,width,Wd,Wdunits),
   get_neg_area(Face,Neg_Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Area is ((New_Ht * New_Wd) - Neg_Area),!.

get_area(Extens,Area,Units) :-
   part_of(Extens,Face),
   dimension(Face,height,Ht,Htunits),
   dimension(Extens,width,Wd,Wdunits),
   get_neg_area(Face,Neg_Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Area is ((New_Ht * New_Wd) - Neg_Area),!.

get_area(Extens,Area,Units) :-
   part_of(Extens,Face),
   dimension(Extens,height,Ht,Htunits),
   dimension(Face,width,Wd,Wdunits),
   get_neg_area(Face,Neg_Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Area is ((New_Ht * New_Wd) - Neg_Area),!.

materials_report :-
   assertz(mat_cost(0)),
   nl,nl,nl,write('        Raw Materials Report'),nl,nl,
   write(' Item          Cost       Units Required'),nl,nl,
   material_list(Material,Num_Units,Item_Cost),
   New_Cost is floor(Item_Cost),
   print_mat_report(Material,Num_Units,New_Cost),
   update_mat_cost(New_Cost),fail.
```

```
materials_report :-
   mat_cost(Total),nl,nl,
   write('************************************'),nl,
   write('*                                  *'),nl,
   write(' Total material cost is $'),
   write(Total),nl,
   write('*                                  *'),nl,
   write('************************************'),nl,nl,nl,
   fail.

update_mat_cost(Item_Cost) :-
   retract(mat_cost(Total)),
   New_Total is (Total + Item_Cost),
   assertz(mat_cost(New_Total)),!.

print_mat_report(Material,Num_Units,Tot_Cost) :-
   write(Material),
   name(Material,L1),
   length(L1,N1),
   tab(17 - N1),
   write('$'),write(Tot_Cost),
   name(Tot_Cost,L2),
   length(L2,N2),
   tab(15 - N2),
   write(Num_Units),nl,nl,!.

report_subst :-
   nl,nl,nl,
   write('  Start Raw Materials Report (w/ substitute)'),
   nl,nl,nl,fail.

report_subst :-
   substitute(Extens,Subst_Mat),
   replace_data(Extens,Subst_Mat),
   not(raw_materials_needed),
   not(materials_report),
   restore_data,fail.

replace_data(Extens,Subst_Mat) :-
   retract(mat_cost(_)),
   retract(material_list(_,_,_)),fail.
```

```
replace_data(Extens,Subst_Mat) :-
   retract(substitute(Extens,Subst_Mat)),
   retract(property(Extens,material_type,Material)),
   write('*****************************************************'),nl,
   write('*                                                   *'),nl,
   write(' '),write(Extens),write(': substitute '),
   write(Subst_Mat),write(' for '),write(Material),nl,
   write('*                                                   *'),nl,
   write('*****************************************************'),nl,nl,
   assertz(property(Extens,material_type,Subst_Mat)),
   assertz(temp(Extens,material_type,Material)),!.

restore_data :-
   retract(temp(Extens,material_type,Material)),
   retract(property(Extens,material_type,_)),
   assertz(property(Extens,material_type,Material)),!.
```

B. STANDARDS DATA

```
minimum(door,door1,width,32,inches).
minimum(door,door1,height,6,feet).
maximum(door,door1,width,4,feet).
maximum(door,door1,height,7,feet).
minimum(door,_,depth,2,inches).
maximum(door,_,depth,3,inches).

minimum(pane,_,quality,3).

comment(masonry,'approved methods must be used for building masonry walls
        when outside air temperature drops below 40 degrees farenheit').
comment_for(cover,brick,masonry).
comment_for(cover,concrete_block,masonry).
comment_for(sub_cover,brick,masonry).
comment_for(sub_cover,concrete_block,masonry).

comment(framing,'grade marks must be clearly visible on all framing members for
        inspection').
comment_for(frame,wood,framing).

check_for(sub_cover,tar_paper,[tar_paper1,tar_paper2,tar_paper3]).
```

C. ASSEMBLY RULES

```
/* start with information on face normals */
assemble(H,house) :-
   assertz(operation(comment,'normal for each face listed',_,_)),
   assertz(operation('FACE','X','Y','Z')),
   assertz(operation('_____','__','__','__')),
   kind_of(Face,face),
   normal_X(Face,X),
   normal_Y(Face,Y),
   normal_Z(Face,Z),
   assertz(operation(Face,X,Y,Z)).

/* start with frame */
assemble(H,house) :-
   assertz(operation(comment,'erect foundation and frame',_,_)).

/* do foundation frame */
assemble(H,house) :-
   kind_of(Yface,face),
   trans_partof(Yface,H),
   normal_Z(Yface,1),
   contains(Yface,L),
   member(Frame,L),
   kind_of(Frame,frame),
   property(Frame,material_type,Mtype),
   assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* do frame perpendicular to ground */
assemble(H,house) :-
   kind_of(Yface,face),
   trans_partof(Yface,H),
   normal_Y(Yface,0),
   normal_Z(Yface,0),
   contains(Yface,L),
   member(Frame,L),
   kind_of(Frame,frame),
   property(Frame,material_type,Mtype),
   assertz(operation(Frame,assemble,'material type: ',Mtype)).
```

```
assemble(H,house) :-
   kind_of(Yface,face),
   trans_partof(Yface,H),
   normal_X(Yface,0),
   normal_Z(Yface,0),
   contains(Yface,L),
   member(Frame,L),
   kind_of(Frame,frame),
   property(Frame,material_type,Mtype),
   assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* ceiling frame */
assemble(H,house) :-
   kind_of(Yface,face),
   trans_partof(Yface,H),
   normal_Z(Yface,-1),
   contains(Yface,L),
   member(Frame,L),
   kind_of(Frame,frame),
   property(Frame,material_type,Mtype),
   assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* roof frame */
assemble(H,house) :-
   kind_of(Roof,roof),
   trans_partof(Roof,H),
   kind_of(Yface,face),
   trans_partof(Yface,Roof),
   contains(Yface,L),
   member(Frame,L),
   kind_of(Frame,frame),
   property(Frame,material_type,Mtype),
   assertz(operation(Frame,assemble,'material type: ',Mtype)).

/* now put doors in place */
assemble(H,house) :-
   assertz(operation(comment,'put door framing in place',_,_)).
```

```
assemble(H,house) :-
   kind_of(Door,door),
   trans_partof(Door,H),
   property(Door,material_type,Mtype),
   assertz(operation(Door,assemble,'material type: ',Mtype)),
   get_faces(Door,Face1,Face2),
   assertz(operation('','- attach to:',Face1,Face2)),
   part_of(Door,Face3),
   assertz(operation('','- location','relative to',Face3)),
   coordinates_X(local,Door,X,Units_X),
   coordinates_Y(local,Door,Y,Units_Y),
   coordinates_Z(local,Door,Z,Units_Z),
   assertz(operation('',' X coordinate',X,Units_X)),
   assertz(operation('',' Y coordinate',Y,Units_Y)),
   assertz(operation('',' Z coordinate',Z,Units_Z)).
```

/* put window sills in place */

```
assemble(H,house) :-
   assertz(operation(comment,'put window framing in place',_,_)).

assemble(H,house) :-
   kind_of(W,window),
   trans_partof(W,H),
   contains(W,L),
   member(Sill,L),
   kind_of(Sill,sill),
   assertz(operation(Sill,assemble,'window sill for: ',W)),
   get_faces(W,Face1,Face2),
   assertz(operation('','- attach to: ',Face1,Face2)),
   part_of(W,Face3),
   assertz(operation('','- location','relative to',Face3)),
   coordinates_X(local,W,X,Units_X),
   coordinates_Y(local,W,Y,Units_Y),
   coordinates_Z(local,W,Z,Units_Z),
   assertz(operation('',' X coordinate',X,Units_X)),
   assertz(operation('',' Y coordinate',Y,Units_Y)),
   assertz(operation('',' Z coordinate',Z,Units_Z)).
```

/* put up exterior siding */

```
assemble(H,house) :-
   assertz(operation(comment,'put up exterior siding',_,_)).
```

```
assemble(H,house) :-
   kind_of(E,exterior),
   trans_partof(E,H),
   contains(E,L),
   assemble(L,face).

/* put up roof */
assemble(H,house) :-
   assertz(operation(comment,'put up roof',_,_)).

assemble(H,house) :-
   kind_of(R,roof),
   trans_partof(R,H),
   contains(R,L),
   assemble(L,face).

/* put up faces for each room */
assemble(H,house) :-
   assertz(operation(comment,'put up faces for each room',_,_)).

assemble(H,house) :-
   kind_of(R,room),
   trans_partof(R,H),
   contains(R,L),
   assemble(L,face).

/* put up windows */
assemble(H,house) :-
   assertz(operation(comment,'put windows in place',_,_)).

assemble(H,house) :-
   kind_of(W,window),
   trans_partof(W,H),
   contains(W,L),
   member(P,L),
   kind_of(P,pane),
   member(C,L),
   kind_of(C,case),
   assertz(operation(W,'complete using',P,C)).
```

```
/* take care of finish on windows and doors */
assemble(H,house) :-
   assertz(operation(comment,'put finish on windows and doors',_,_)).

assemble(H,house) :-
   finish.

/* take care of door knobs and hinges */
assemble(H,house) :-
   assertz(operation(comment,'put on door knobs and hinges',_,_)).

assemble(H,house) :-
   kind_of(D,door),
   trans_partof(D,H),
   assemble(D,door).

/* take care of paint on faces */
assemble(H,house) :-
   assertz(operation(comment,'put final paint on faces',_,_)).

assemble(H,house) :-
   kind_of(R,roof),
   trans_partof(R,H),
   contains(R,L),
   paint_face(L).

assemble(H,house) :-
   kind_of(E,exterior),
   trans_partof(E,H),
   contains(E,L),
   paint_face(L).

assemble(H,house) :-
   kind_of(R,room),
   trans_partof(R,H),
   contains(R,L),
   paint_face(L).
```

```
/* routines to put up sub_covers and covers for a given    */
/* list of faces supplied as first argument; these routines */
/* look for common materials to help set priority; all     */
/* sub_covers are handled prior to covers;                 */
/* covers which are paint are left to be performed at a    */
/* later time; all sub_covers and covers                   */
/* associated with the floor are performed last            */

assemble(L,face) :-
   assemble1(L,[],face).

assemble(L,face) :-
   member(Face,L),
   normal_Z(Face,1),
   assertz(operation(comment,'build floor as last step',_,_)),
   contains(Face,L1),
   assemble2([L1],[L1],face).

assemble1(L,L1,face) :-
   member(Face,L),
   not(normal_Z(Face,1)),
   delete(Face,L,L2),
   contains(Face,L3),
   assemble1(L2,[L3|L1],face),!.

assemble1(L,L1,face) :-
   assemble2(L1,L1,face),!.

assemble2(Full_L,L,face) :-
   member(Face,L),
   delete(Face,L,L1),
   member(Item,Face),
   kind_of(Item,sub_cover),
   property(Item,material_type,Mtype),
   operation(Y,_,_,Mtype),
   member(Face1,Full_L),
   member(Y,Face1),
   assertz(operation(Item,assemble,'material type: ',Mtype)),
   delete(Item,Face,Face2),
   assemble2(Full_L,[Face2|L1],face),!.
```

```
assemble2(Full_L,L,face) :-
   member(Face,L),
   delete(Face,L,L1),
   member(Item,Face),
   kind_of(Item,sub_cover),
   property(Item,material_type,Mtype),
   assertz(operation(Item,assemble,'material type: ',Mtype)),
   delete(Item,Face,Face1),
   assemble2(Full_L,[Face1|L1],face),!.

assemble2(Full_L,L,face) :-
   member(Face,L),
   delete(Face,L,L1),
   member(Item,Face),
   kind_of(Item,cover),
   property(Item,material_type,Mtype),
   not(liquid(Mtype,paint,_,_,_,_,_)),
   operation(Y,_,_,Mtype),
   member(Face1,Full_L),
   member(Y,Face1),
   assertz(operation(Item,assemble,'material type: ',Mtype)),
   delete(Item,Face,Face2),
   assemble2(Full_L,[Face2|L1],face),!.

assemble2(Full_L,L,face) :-
   member(Face,L),
   delete(Face,L,L1),
   member(Item,Face),
   kind_of(Item,cover),
   property(Item,material_type,Mtype),
   not(liquid(Mtype,paint,_,_,_,_,_)),
   assertz(operation(Item,assemble,'material type: ',Mtype)),
   delete(Item,Face,Face1),
   assemble2(Full_L,[Face1|L1],face),!.

assemble2(Full_L,L,face).

/* take care of finishes */
finish :-
   property(F,finish_type,Ftype),
   property(F,finish_color,Fcolor),
   assertz(operation(F,finish,Ftype,Fcolor)).
```

```
/* assemble door knob */
assemble(D,door) :-
   property(D,knob_type,Ktype),
   assertz(operation(D,assemble,knob,Ktype)).

/* assemble door hinges */
assemble(D,door) :-
   property(D,hinge_type,Htype),
   assertz(operation(D,assemble,hinge,Htype)).

/* routines to apply paint to faces; acts on covers only */
paint_face(L) :-
   member(Face,L),
   normal_Z(Face,-1),
   contains(Face,L1),
   member(Cover,L1),
   kind_of(Cover,cover),
   property(Cover,material_type,Mtype),
   liquid(Mtype,paint,_,_,_,_,_),
   assertz(operation(Cover,paint,'material type: ',Mtype)),
   delete(Face,L,L2),
   paint_face(L2),!.

paint_face(L) :-
   member(Face,L),
   normal_Y(Face,0),
   normal_Z(Face,0),
   contains(Face,L1),
   member(Cover,L1),
   kind_of(Cover,cover),
   property(Cover,material_type,Mtype),
   liquid(Mtype,paint,_,_,_,_,_),
   assertz(operation(Cover,paint,'material type: ',Mtype)),
   delete(Face,L,L2),
   paint_face(L2),!.
```

```
paint_face(L) :-
   member(Face,L),
   normal_X(Face,0),
   normal_Z(Face,0),
   contains(Face,L1),
   member(Cover,L1),
   kind_of(Cover,cover),
   property(Cover,material_type,Mtype),
   liquid(Mtype,paint,_,_,_,_,_),
   assertz(operation(Cover,paint,'material type: ',Mtype)),
   delete(Face,L,L2),
   paint_face(L2),!.

paint_face(L) :-
   member(Face,L),
   normal_Z(Face,-1),
   contains(Face,L1),
   member(Cover,L1),
   kind_of(Cover,cover),
   property(Cover,material_type,Mtype),
   liquid(Mtype,paint,_,_,_,_,_),
   assertz(operation(Cover,paint,'material type: ',Mtype)),
   delete(Face,L,L2),
   paint_face(L2),!.

paint_face(L) :-
   member(Face,L),
   contains(Face,L1),
   member(Cover,L1),
   kind_of(Cover,cover),
   property(Cover,material_type,Mtype),
   liquid(Mtype,paint,_,_,_,_,_),
   assertz(operation(Cover,paint,'material type: ',Mtype)),
   delete(Face,L,L2),
   paint_face(L2),!.

paint_face(L).
```

```
/* routine to get the two faces which an item is associated with */
get_faces(Item,Face1,Face2) :-
   face(Item,Face1),
   face(Item,Face2),
   not(Face1 = Face2),!.
```

D. BILL OF MATERIALS RULES

```
/* materials for doors */
raw_materials_needed :-
   kind_of(Extens,door),
   material(Extens,_,_,_,_,_,_,_,_,_,_,Cost),
   add_material(Extens,1,Cost),fail.

raw_materials_needed :-
   kind_of(Extens,door),
   property(Extens,finish_type,Paint),
   liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
   dimension(Extens,height,Org_Ht,Ht_Units),
   dimension(Extens,width,Org_Wd,Wd_Units),
   dimension(Extens,depth,Org_Dp,Dp_Units),
   convert(Org_Ht,Ht_Units,New_Ht,Area_Units),
   convert(Org_Wd,Wd_Units,New_Wd,Area_Units),
   convert(Org_Dp,Dp_Units,New_Dp,Area_Units),
   Area is ((2 * New_Ht * New_Wd) + (2 * New_Ht * New_Dp) +
   (2 * New_Wd * New_Dp)),
   Num_Units is (Area / Area_Cov),
   Tot_Cost is (Num_Units * Cost),
   add_material(Paint,Num_Units,Tot_Cost),fail.

/* materials for windows */
raw_materials_needed :-
   kind_of(Extens,window),
   material(Extens,_,_,_,_,_,_,_,_,_,_,Cost),
   add_material(Extens,1,Cost),fail.
```

```
raw_materials_needed :-
   kind_of(Window,window),
   part_of(Extens,Window),
   kind_of(Extens,sill),
   property(Extens,finish_type,Paint),
   liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
   convert(Area_Cov,Area_Units,New_Area,feet),
   New_Area2 is ((New_Area * New_Area) / Area_Cov),
   dimension(Window,height,Org_Ht,Ht_Units),
   dimension(Window,width,Org_Wd,Wd_Units),
   part_of(Window,Face),
   dimension(Face,depth,Org_Dp,Dp_Units),
   convert(Org_Ht,Ht_Units,New_Ht,Area_Units),
   convert(Org_Wd,Wd_Units,New_Wd,Area_Units),
   convert(Org_Dp,Dp_Units,New_Dp,Area_Units),
   Area is ((2 * New_Ht * New_Dp) + (2 * New_Wd * New_Dp)),
   Num_Units is (Area / Area_Cov),
   Tot_Cost is (Num_Units * Cost),
   add_material(Paint,Num_Units,Tot_Cost),fail.

/* materials for frames; assumes 1 square foot of area */
/* requires a 1 foot length of frame wood              */
raw_materials_needed :-
   kind_of(Extens,frame),
   dimension(Extens,height,Height,Ht_Units),
   dimension(Extens,width,Width,Wd_Units),
   convert(Height,Ht_Units,New_Height,feet),
   convert(Width,Wd_Units,New_Width,feet),
   property(Extens,material_type,Material),
   material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
   convert(Len,Lenunits,New_Len,feet),
   Area is (New_Height * New_Width),
   Num_Units is (Area / New_Len),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
   kind_of(Extens,frame),
   not(dimension(Extens,width,_,_)),
   not(dimension(Extens,height,_,_)),
   property(Extens,material_type,Material),
   material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Area,Units,New_Area,feet),
   New_Area2 is ((New_Area * New_Area) / Area),
   longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
   convert(Len,Lenunits,New_Len,feet),
   Num_Units is (Area / Len),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
   normal_Z(Face,-1),
   part_of(Extens,Face),
   kind_of(Extens,frame),
   property(Extens,material_type,Material),
   material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
   convert(Len,Lenunits,New_Len,feet),
   kind_of(Roof,roof),
   part_of(Face2,Roof),
   kind_of(Face2,face),
   part_of(Extens2,Face2),
   kind_of(Extens2,frame),
   normal_Z(Face2,CosZ),
   dimension(Extens2,height,Ht_face,Ht_face_units),
   convert(Ht_face,Ht_face_units,New_Ht_face,feet),
   dimension(Extens2,width,Wd_face,Wd_face_units),
   convert(Wd_face,Wd_face_units,New_Wd_face,feet),
   SinZ is (sqrt(1 - (CosZ * CosZ))),
   Area is (SinZ * New_Ht_face * New_Wd_face),
   Area2 is (SinZ * New_Ht_face * CosZ * New_Ht_face * 2),
   Tot_Area is Area + Area2,
   Num_Units is (Tot_Area / New_Len),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
   normal_Z(Face,-1),
   part_of(Extens,Face),
   kind_of(Extens,frame),
   property(Extens,material_type,Material),
   material(Material,wood,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Lenunits),
   convert(Len,Lenunits,New_Len,feet),
   kind_of(Roof,roof),
   part_of(Face2,Roof),
   kind_of(Face2,face),
   part_of(Extens2,Face2),
   kind_of(Extens2,frame),
   normal_Z(Face2,CosZ),
   not(dimension(Extens2,height,_,_)),
   dimension(Face2,height,Ht_face,Ht_face_units),
   convert(Ht_face,Ht_face_units,New_Ht_face,feet),
   dimension(Face2,width,Wd_face,Wd_face_units),
   convert(Wd_face,Wd_face_units,New_Wd_face,feet),
   SinZ is (sqrt(1 - (CosZ * CosZ))),
   Area is (SinZ * New_Ht_face * New_Wd_face),
   Area2 is (SinZ * New_Ht_face * CosZ * New_Wd_face),
   Tot_Area is Area + Area2,
   Num_Units is (Tot_Area / New_Len),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.
```

/* frame material of type "filler" */

```
raw_materials_needed :-
   kind_of(Extens,frame),
   property(Extens,material_type,Material),
   filler(Material,_,Vol,Volunits,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Vol,Volunits,New_Vol,Units),
   New_Vol2 is ((New_Vol * New_Vol * New_Vol)/(Vol * Vol)),
   dimension(Extens,depth,Dp,Dpunits),
   convert(Dp,Dpunits,New_Dp,Units),
   Num_Units is (Area * New_Dp / New_Vol2),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
   kind_of(Extens,sub_cover),
   dimension(Extens,depth,Th,Thunits),
   property(Extens,material_type,Material),
   material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   match(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Th,Thunits,Act_Ht,Units1,Act_Wd,Units2),
   get_area(Extens,Area,Units),
   convert(Act_Ht,Units1,Act_Ht2,Units),
   convert(Act_Wd,Units2,Act_Wd2,Units),
   Num_Units is (Area / (Act_Ht2 * Act_Wd2)),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
   kind_of(Extens,sub_cover),
   not(dimension(Extens,depth,Th,Thunits)),
   property(Extens,material_type,Material),
   material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Num_Units is (Area / (New_Ht * New_Wd)),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
   kind_of(Extens,sub_cover),
   property(Extens,material_type,Paint),
   liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Area_Cov,Area_Units,New_Area,Units),
   New_Area2 is ((New_Area * New_Area) / Area_Cov),
   Num_Units is (Area / New_Area2),
   Tot_Cost is (Num_Units * Cost),
   add_material(Paint,Num_Units,Tot_Cost),fail.
```

```
raw_materials_needed :-
   kind_of(Extens,cover),
   dimension(Extens,depth,Th,Thunits),
   property(Extens,material_type,Material),
   material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   match(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Th,Thunits,Act_Ht,Units1,Act_Wd,Units2),
   get_area(Extens,Area,Units),
   convert(Act_Ht,inches,Act_Ht2,Units),
   convert(Act_Wd,inches,Act_Wd2,Units),
   Num_Units is (Area / (Act_Ht2 * Act_Wd2)),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
   kind_of(Extens,cover),
   not(dimension(Extens,depth,Th,Thunits)),
   property(Extens,material_type,Material),
   material(Material,_,Ht,Htunits,Wd,Wdunits,Dp,Dpunits,_,_,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Ht,Htunits,New_Ht,Units),
   convert(Wd,Wdunits,New_Wd,Units),
   Num_Units is (Area / (New_Ht * New_Wd)),
   Tot_Cost is (Num_Units * Cost),
   add_material(Material,Num_Units,Tot_Cost),fail.

raw_materials_needed :-
   kind_of(Extens,cover),
   property(Extens,material_type,Paint),
   liquid(Paint,_,Area_Cov,Area_Units,_,_,Cost),
   get_area(Extens,Area,Units),
   convert(Area_Cov,Area_Units,New_Area,Units),
   New_Area2 is ((New_Area * New_Area) / Area_Cov),
   Num_Units is (Area / New_Area2),
   Tot_Cost is (Num_Units * Cost),
   add_material(Paint,Num_Units,Tot_Cost),fail.

add_material(Material,Num_Units,Tot_Cost) :-
   retract(material_list(Material,Old_Num_Units,Old_Cost)),
   New_Num_Units is (Old_Num_Units + Num_Units),
   New_Cost is (Old_Cost + Tot_Cost),
   assertz(material_list(Material,New_Num_Units,New_Cost)),!.
```

```
add_material(Material,Num_Units,Tot_Cost):-
   assertz(material_list(Material,Num_Units,Tot_Cost)),!.

/* material data */

material(shingle12,shingle,12,inches,6,inches,0.25,inches,0,feet,0,feet,1.25).

material(tar_paper2,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,feet,125.00).

material(tar_paper1,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,feet,150.00).

material(tar_paper3,tar_paper,72,inches,240,inches,0.25,inches,0,feet,0,feet,110.00).

material(sheath_paper24,sheath_paper,12,feet,100,feet,0.1,inches,0,feet,0,feet,75.65).

material(wood8,wood,144,inches,4,inches,2,inches,0,feet,0,feet,8.25).

material(hard_wood9,hard_wood,4,inch,24,feet,0.5,inch,0,feet,0,feet,12.00).

material(hardboard32,hardboard,36,feet,10,feet,1,inches,0,feet,0,feet,136.55).

material(hardboard78,hardboard,36,feet,24,feet,1,inch,0,feet,0,feet,289.00).

material(hardboard34,hardboard,24,feet,10,feet,1,inches,0,feet,0,feet,95.35).

/* use brick 10x4x6 effective size */
material(brick88,brick,10,inches,4,inches,6,inches,0,feet,0,feet,1.15).

liquid(paint9,paint,900,feet,1,gallon,8.00).

liquid(paint21,paint,700,feet,1,gallon,13.55).

liquid(paint17,paint,1100,feet,1,gallon,8.25).

/* 10 lb per 2 cubic feet */
filler(concrete1,concrete,2,feet,10,lb,5.00).

material(door1,_,_,_,_,_,_,_,0,feet,0,feet,16.00).

material(window1,_,_,_,_,_,_,_,0,feet,0,feet,30.50).
```

E. DESIGN DATA

```
/*****************************************************/

/* house data */

kind_of(house1,house).

property(house1,subtype,single_room).

contains(house1,[roof1,exterior1,room1]).

/*****************************************************/

/* exterior data */

kind_of(exterior1,exterior).

contains(exterior1,[face5,face6,face7,face8]).
part_of(exterior1,house1).

/*****************************************************/

/* roof data */

kind_of(roof1,roof).

contains(roof1,[face11,face12]).
part_of(roof1,house1).

/*****************************************************/
```

```
kind_of(face11,face).

dimension(face11,height,151.5,inches).
dimension(face11,width,384,inches).
dimension(face11,depth,6.5,inches).

contains(face11,[frame1,sub_cover2,sub_cover1,cover1]).
normal_X(face11,0).
normal_Y(face11,0.34).
normal_Z(face11,0.94).
part_of(face11,roof1).

/*-----------------------------------------------*/

kind_of(frame1,frame).

property(frame1,material_type,wood8).

dimension(frame1,height,139.5,inches).
dimension(frame1,width,382,inches).
dimension(frame1,depth,4,inches).

face(frame1,face11).

part_of(frame1,face11).

/*-----------------------------------------------*/

kind_of(sub_cover2,sub_cover).

property(sub_cover2,material_type,wood8).

dimension(sub_cover2,depth,2,inches).

part_of(sub_cover2,face11).

/*-----------------------------------------------*/
```

```
kind_of(sub_cover1,sub_cover).

property(sub_cover1,material_type,tar_paper2).

dimension(sub_cover1,depth,0.25,inches).

part_of(sub_cover1,face11).

/*-----------------------------------------------*/

kind_of(cover1,cover).

property(cover1,material_type,shingle12).
property(cover1,finish_color,brown).

dimension(cover1,depth,0.25,inches).

part_of(cover1,face11).

/*************************************************/

kind_of(face12,face).

dimension(face12,height,151.5,inches).
dimension(face12,width,384,inches).
dimension(face12,depth,6.5,inches).

contains(face12,[frame2,sub_cover13,sub_cover14,cover12]).
normal_X(face12,0).
normal_Y(face12,-0.34).
normal_Z(face12,0.94).
part_of(face12,roof1).

/*-----------------------------------------------*/
```

```
kind_of(frame2,frame).

property(frame2,material_type,wood8).

dimension(frame2,height,139.5,inches).
dimension(frame2,width,382,inches).
dimension(frame2,depth,4,inches).

face(frame2,face12).

part_of(frame2,face12).

/*-----------------------------------------------*/

kind_of(sub_cover13,sub_cover).

property(sub_cover13,material_type,wood8).

dimension(sub_cover13,depth,2,inches).

part_of(sub_cover13,face12).

/*-----------------------------------------------*/

kind_of(sub_cover14,sub_cover).

property(sub_cover14,material_type,tar_paper2).

dimension(sub_cover14,depth,0.25,inches).

part_of(sub_cover14,face12).

/*-----------------------------------------------*/
```

```
kind_of(cover12,cover).

property(cover12,material_type,shingle12).
property(cover12,finish_color,brown).

dimension(cover12,depth,0.25,inches).

part_of(cover12,face12).

/*******************************************************/

/* room1 */

kind_of(room1,room).

coordinates_X(product,room1,0,inches).
coordinates_Y(product,room1,0,inches).
coordinates_Z(product,room1,12,inches).
contains(room1,[face1,face2,face3,face4,face9,face10]).
part_of(room1,house1).

/*******************************************************/

/* face1 */

kind_of(face1,face).

dimension(face1,height,115,inches).
dimension(face1,width,362,inches).
dimension(face1,depth,1,inches).

contains(face1,[sub_cover3,cover2]).
normal_X(face1,0).
normal_Y(face1,-1).
normal_Z(face1,0).
part_of(face1,room1).

/*-----------------------------------------------*/
```

```
kind_of(sub_cover3,sub_cover).

property(sub_cover3,material_type,hardboard32).

dimension(sub_cover3,depth,1,inches).

part_of(sub_cover3,face1).

/*-----------------------------------------*/

kind_of(cover2,cover).

property(cover2,material_type,paint9).
property(cover2,finish_color,yellow).

part_of(cover2,face1).

/*************************************************/

/* face2 */

kind_of(face2,face).

dimension(face2,height,115,inches).
dimension(face2,width,240,inches).
dimension(face2,depth,1,inches).

contains(face2,[sub_cover4,cover3]).
normal_X(face2,-1).
normal_Y(face2,0).
normal_Z(face2,0).
part_of(face2,room1).

/*-----------------------------------------*/
```

```
kind_of(sub_cover4,sub_cover).

property(sub_cover4,material_type,hardboard34).

dimension(sub_cover4,depth,1,inches).

part_of(sub_cover4,face2).

/*-----------------------------------------*/

kind_of(cover3,cover).

property(cover3,material_type,paint9).
property(cover3,finish_color,yellow).

part_of(cover3,face2).

/*************************************************/

/* face3 */

kind_of(face3,face).

dimension(face3,height,115,inches).
dimension(face3,width,362,inches).
dimension(face3,depth,1,inches).

contains(face3,[sub_cover5,cover4]).
normal_X(face3,0).
normal_Y(face3,1).
normal_Z(face3,0).
part_of(face3,room1).

/*-----------------------------------------*/
```

```
kind_of(sub_cover5,sub_cover).

property(sub_cover5,material_type,hardboard32).

dimension(sub_cover5,depth,1,inches).

part_of(sub_cover5,face3).

/*-----------------------------------------------*/

kind_of(cover4,cover).

property(cover4,material_type,paint9).
property(cover4,finish_color,yellow).

part_of(cover4,face3).

/*************************************************/

/* face4 */

kind_of(face4,face).

dimension(face4,height,115,inches).
dimension(face4,width,240,inches).
dimension(face4,depth,1,inches).

contains(face4,[sub_cover6,cover5]).
normal_X(face4,1).
normal_Y(face4,0).
normal_Z(face4,0).
part_of(face4,room1).

/*-----------------------------------------------*/
```

```
kind_of(sub_cover6,sub_cover).

property(sub_cover6,material_type,hardboard34).

dimension(sub_cover6,depth,1,inches).

part_of(sub_cover6,face4).

/*-----------------------------------------------*/

kind_of(cover5,cover).

property(cover5,material_type,paint9).
property(cover5,finish_color,yellow).

part_of(cover5,face4).

/***********************************************/

/* face5 */
/* use brick 10x4x6 effective size */

kind_of(face5,face).

dimension(face5,height,120,inches).
dimension(face5,width,382,inches).
dimension(face5,depth,6,inches).

contains(face5,[frame3,sub_cover7,cover6]).
normal_X(face5,0).
normal_Y(face5,1).
normal_Z(face5,0).
part_of(face5,exterior1).

/*-----------------------------------------------*/
```

```
kind_of(frame3,frame).

property(frame3,material_type,wood8).

dimension(frame3,depth,4,inches).

face(frame3,face5).
face(frame3,face1).

part_of(frame3,face5).

/*-----------------------------------------------*/

kind_of(sub_cover7,sub_cover).

property(sub_cover7,material_type,sheath_paper24).

part_of(sub_cover7,face5).

/*-----------------------------------------------*/

kind_of(cover6,cover).

property(cover6,material_type,brick88).
property(cover6,finish_color,red).

dimension(cover6,depth,6,inches).

part_of(cover6,face5).
```

```
/*************************************************/

/* face6 */

kind_of(face6,face).

dimension(face6,height,120,inches).
dimension(face6,width,250,inches).
dimension(face6,depth,6,inches).

contains(face6,[frame4,sub_cover8,cover7,window1]).
normal_X(face6,1).
normal_Y(face6,0).
normal_Z(face6,0).
part_of(face6,exterior1).

/*-----------------------------------------------*/

kind_of(frame4,frame).

property(frame4,material_type,wood8).

dimension(frame4,depth,4,inches).

face(frame4,face6).
face(frame4,face2).

part_of(frame4,face6).

/*-----------------------------------------------*/

kind_of(sub_cover8,sub_cover).

property(sub_cover8,material_type,sheath_paper24).

part_of(sub_cover8,face6).

/*-----------------------------------------------*/
```

```
kind_of(cover7,cover).

property(cover7,material_type,brick88).
property(cover7,finish_color,red).

dimension(cover7,depth,6,inches).

part_of(cover7,face6).

/*-----------------------------------------------*/

kind_of(window1,window).

dimension(window1,height,36,inches).
dimension(window1,width,48,inches).
dimension(window1,depth,0.5,inches).

contains(window1,[pane1,sill1,case1]).
face(window1,face2).
face(window1,face6).
coordinates_X(local,window1,96,inches).
coordinates_Y(local,window1,0,inches).
coordinates_Z(local,window1,66,inches).
part_of(window1,face6).

/*-----------------------------------------------*/

kind_of(pane1,pane).

property(pane1,quality,4).

part_of(pane1,window1).

/*-----------------------------------------------*/
```

```
kind_of(sill1,sill).

property(sill1,finish_type,paint17).
property(sill1,finish_color,white).

part_of(sill1,window1).

/*-----------------------------------------------*/

kind_of(case1,case).

part_of(case1,window1).

/*************************************************/

/* face7 */

kind_of(face7,face).

dimension(face7,height,120,inches).
dimension(face7,width,382,inches).
dimension(face7,depth,6,inches).

contains(face7,[frame5,sub_cover9,cover8,door1]).
normal_X(face7,0).
normal_Y(face7,-1).
normal_Z(face7,0).
part_of(face7,exterior1).

/*-----------------------------------------------*/

kind_of(frame5,frame).

property(frame5,material_type,wood8).

dimension(frame5,depth,4,inches).

face(frame5,face7).
face(frame5,face3).

part_of(frame5,face7).
```

```
/*-----------------------------------------------*/

kind_of(sub_cover9,sub_cover).

property(sub_cover9,material_type,sheath_paper24).

part_of(sub_cover9,face7).

/*-----------------------------------------------*/

kind_of(cover8,cover).

property(cover8,material_type,brick88).
property(cover8,finish_color,red).

dimension(cover8,depth,6,inches).

part_of(cover8,face7).

/*-----------------------------------------------*/

kind_of(door1,door).

property(door1,material_type,wood5).
property(door1,finish_type,paint21).
property(door1,finish_color,brown).
property(door1,knob_type,round32).
property(door1,hinge_type,square3in).

dimension(door1,height,84,inches).
dimension(door1,width,36,inches).
dimension(door1,depth,2.5,inches).

face(door1,face3).
face(door1,face7).
coordinates_X(local,door1,125,inches).
coordinates_Y(local,door1,0,inches).
coordinates_Z(local,door1,42,inches).
part_of(door1,face7).
```

```
/*************************************************/

/* face8 */

kind_of(face8,face).

dimension(face8,height,120,inches).
dimension(face8,width,250,inches).
dimension(face8,depth,6,inches).

contains(face8,[frame6,sub_cover10,cover9]).
normal_X(face8,-1).
normal_Y(face8,0).
normal_Z(face8,0).
part_of(face8,exterior1).

/*-----------------------------------------------*/

kind_of(frame6,frame).

property(frame6,material_type,wood8).

dimension(frame6,depth,4,inches).

face(frame6,face8).
face(frame6,face4).

part_of(frame6,face8).

/*-----------------------------------------------*/

kind_of(sub_cover10,sub_cover).

property(sub_cover10,material_type,sheath_paper24).

part_of(sub_cover10,face8).

/*-----------------------------------------------*/
```

```
kind_of(cover9,cover).

property(cover9,material_type,brick88).
property(cover9,finish_color,red).

part_of(cover9,face8).

/*****************************************************/

/* face9 */

kind_of(face9,face).

dimension(face9,height,20,feet).
dimension(face9,width,30,feet).
dimension(face9,depth,1,inches).

contains(face9,[frame7,sub_cover11,cover10]).
normal_X(face9,0).
normal_Y(face9,0).
normal_Z(face9,-1).
part_of(face9,room1).

/*-----------------------------------------------*/

kind_of(frame7,frame).

property(frame7,material_type,wood8).

face(frame7,face9).

part_of(frame7,face9).

/*-----------------------------------------------*/
```

```
kind_of(sub_cover11,sub_cover).

property(sub_cover11,material_type,hardboard78).

dimension(sub_cover11,depth,1,inches).

part_of(sub_cover11,face9).

/*-----------------------------------------------*/

kind_of(cover10,cover).

property(cover10,material_type,paint17).
property(cover10,finish_color,white).

part_of(cover10,face9).

/*************************************************/

/* face10 */

kind_of(face10,face).

dimension(face10,height,382,inches).
dimension(face10,width,262,inches).
dimension(face10,depth,12.5,inches).

contains(face10,[frame8,sub_cover12,cover11]).
normal_X(face10,0).
normal_Y(face10,0).
normal_Z(face10,1).
part_of(face10,room1).

/*-----------------------------------------------*/
```

```
kind_of(frame8,frame).

property(frame8,material_type,concrete1).

dimension(frame8,depth,12,inches).

face(frame8,face10).

part_of(frame8,face10).

/*-----------------------------------------*/

kind_of(sub_cover12,sub_cover).

property(sub_cover12,material_type,hard_wood9).

dimension(sub_cover12,height,20,feet).
dimension(sub_cover12,width,30,feet).
dimension(sub_cover12,depth,0.5,inches).

part_of(sub_cover12,face10).

/*-----------------------------------------*/

kind_of(cover11,cover).

property(cover11,material_type,paint21).
property(cover11,finish_color,brown).

dimension(cover11,height,20,feet).
dimension(cover11,width,30,feet).

part_of(cover11,face10).
```

F. SCHEMA DATA

```
part_of(house,floorplan).
part_of(house,exterior).
part_of(house,room).
part_of(house,roof).
part_of(house,space).

part_of(roof,face).

part_of(room,face).

part_of(space,face).

part_of(exterior,face).

part_of(face,door).
part_of(face,window).
part_of(face,opening).
part_of(face,covering).
part_of(face,sub_covering).
part_of(face,frame).
part_of(face,insulation).
part_of(face,connection).

part_of(connection,plumbing).
part_of(connection,electric).
part_of(connection,heating).
part_of(connection,gas).

part_of(window,sill).
part_of(window,case).
part_of(window,pane).

trans_partof(X,Y) :- part_of(X,Y),!.
trans_partof(X,Y) :- part_of(X,Z),
   trans_partof(Z,Y),!.
```

G. CONVERSION RULES

```
converts(A,feet,B,feet) :- B = A.
converts(A,inches,B,inches) :- B = A.
converts(A,feet,B,inches) :- B = A * 12.
converts(A,inches,B,feet) :- B = A / 12.
converts(A,feet,B,yards) :- B = A / 3.
converts(A,yards,B,feet) :- B = A * 3.

convert(A,Dimension1,B,Dimension2) :-
converts(A,Dimension1,B,Dimension2),!.

convert(A,Dimension1,B,Dimension2) :-
converts(A,Dimension1,X,Dimensionx),
not(equal(Dimension1,Dimensionx)),
convert(X,Dimensionx,B,Dimension2).
```

H. MISCELLANEOUS ROUTINES

```
/* find longest dimension of three passed in */
longest_dimension(Ht,Htunits,Wd,Wdunits,Dp,Dpunits,Len,Htunits):-
   convert(Wd,Wdunits,New_Wd,Htunits),
   convert(Dp,Dpunits,New_Dp,Htunits),
   maximum(New_Ht,New_Wd,Max),
   maximum(Max,New_Dp,Len),!.

maximum(A,B,A) :-
   A > B,!.

maximum(A,B,B).

/* have match if within .25 inches */
match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,B,B_Units):-
   convert(A,A_Units,New_A,inches),
   convert(B,B_Units,New_B,inches),
   convert(C,C_Units,New_C,inches),
   convert(D,D_Units,New_D,inches),
   ((New_D - New_C) < 0.25),
   ((New_D - New_C) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,C,C_Units):-
   convert(A,A_Units,New_A,inches),
   convert(B,B_Units,New_B,inches),
   convert(C,C_Units,New_C,inches),
   convert(D,D_Units,New_D,inches),
   ((New_D - New_B) < 0.25),
   ((New_D - New_B) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,B,B_Units,C,C_Units):-
   convert(A,A_Units,New_A,inches),
   convert(B,B_Units,New_B,inches),
   convert(C,C_Units,New_C,inches),
   convert(D,D_Units,New_D,inches),
   ((New_D - New_A) < 0.25),
   ((New_D - New_A) > - 0.25),!.

match(A,A_Units,B,B_Units,C,C_Units,D,D_Units,A,A_Units,B,B_Units):-
   nl,write('Error! No match found during raw material calculations.'),fail.
```

```
/* routine to get member of list */
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).

/* routine to delete member of list */
delete(X,[],[]).
delete(X,[X|L],L) :- !.
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

equal(A,B) :- B = A.
```

APPENDIX B - SAMPLE TRANSLATOR EXECUTION

```
% prolog

C-Prolog version 1.5

| ?- [main].

assembly reconsulted 9104 bytes 2.01667 sec.
conversion reconsulted 724 bytes 0.183334 sec.
interface reconsulted 12128 bytes 2.6 sec.
schema reconsulted 1008 bytes 0.266667 sec.
standards reconsulted 680 bytes 0.2 sec.
house1 reconsulted 11956 bytes 3.68333 sec.
routines reconsulted 1736 bytes 0.416674 sec.
materials reconsulted 8584 bytes 2.36667 sec.
main consulted 45920 bytes 11.95 sec.

yes

| ?- start.

check for house house1

check for exterior exterior1

check for roof roof1

check for face face11

check for frame frame1
   grade marks must be clearly visible on all framing members for inspection

check for sub_cover sub_cover2
```

check for sub_cover sub_cover1
  sub_cover sub_cover1 meets requirements; allowed substitutes are:
  - tar_paper1
  - tar_paper3

check for cover cover1

check for face face12

check for frame frame2
  grade marks must be clearly visible on all framing
  members for inspection

check for sub_cover sub_cover13

check for sub_cover sub_cover14
  sub_cover sub_cover14 meets requirements; allowed substitutes are:
  - tar_paper1
  - tar_paper3

check for cover cover12

check for room room1

check for face face1

check for sub_cover sub_cover3

check for cover cover2

check for face face2

check for sub_cover sub_cover4

check for cover cover3

check for face face3

check for sub_cover sub_cover5

check for cover cover4

check for face face4

check for sub_cover sub_cover6

check for cover cover5

check for face face5

check for frame frame3
  grade marks must be clearly visible on all framing
  members for inspection

check for sub_cover sub_cover7

check for cover cover6
  approved methods must be used for building masonry walls
  when outside air temperature drops below 40 degrees farenheit

check for face face6

check for frame frame4
  grade marks must be clearly visible on all framing
  members for inspection

check for sub_cover sub_cover8

check for cover cover7
  approved methods must be used for building masonry walls
  when outside air temperature drops below 40 degrees farenheit

check for window window1

check for pane pane1

  pane pane1 passed quality check

check for sill sill1

check for case case1

check for face face7

check for frame frame5
  grade marks must be clearly visible on all framing members for inspection

check for sub_cover sub_cover9

check for cover cover8
  approved methods must be used for building masonry walls
  when outside air temperature drops below 40 degrees farenheit

check for door door1
  door door1 passed - height
  door door1 passed - width
  door door1 passed - depth

check for face face8

check for frame frame6
  grade marks must be clearly visible on all framing
  members for inspection

check for sub_cover sub_cover10

check for cover cover9
  approved methods must be used for building masonry walls
  when outside air temperature drops below 40 degrees farenheit

check for face face9

check for frame frame7
  grade marks must be clearly visible on all framing members for inspection

check for sub_cover sub_cover11

check for cover cover10

check for face face10

check for frame frame8

check for sub_cover sub_cover12

check for cover cover11

```
**********************************************
 Production Sequence Report for house1
  - house style is single_room
    and consists of [roof1,exterior1,room1]
**********************************************
 comment : normal for each face listed
**********************************************
FACE        X          Y          Z
_____       __         __         __
face11      0          0.34       0.94
face12      0          -0.34      0.94
face1       0          -1         0
face2       -1         0          0
face3       0          1          0
face4       1          0          0
face5       0          1          0
face6       1          0          0
face7       0          -1         0
face8       -1         0          0
face9       0          0          -1
face10      0          0          1
**********************************************
*                              *
 comment : erect foundation and frame
*                              *
**********************************************
frame8      assemble     material type:  concrete1

frame4      assemble     material type:  wood8

frame6      assemble     material type:  wood8

frame3      assemble     material type:  wood8

frame5      assemble     material type:  wood8

frame7      assemble     material type:  wood8

frame1      assemble     material type:  wood8

frame2      assemble     material type:  wood8
```

```
**********************************************
comment : put door framing in place
**********************************************

door1       assemble     material type:  wood5

          - attach to:  face3         face7

          - location    relative to    face7

           X coordinate  125          inches

           Y coordinate  0            inches

           Z coordinate  42           inches


**********************************************
comment : put window framing in place
**********************************************

sill1       assemble     window sill for: window1

          - attach to:  face2         face6

          - location    relative to    face6

           X coordinate  96           inches

           Y coordinate  0            inches

           Z coordinate  66           inches
```

```
**********************************************
comment : put up exterior siding
**********************************************

sub_cover10   assemble     material type:   sheath_paper24

sub_cover9    assemble     material type:   sheath_paper24

sub_cover8    assemble     material type:   sheath_paper24

sub_cover7    assemble     material type:   sheath_paper24

cover6        assemble     material type:   brick88

cover7        assemble     material type:   brick88

cover8        assemble     material type:   brick88

cover9        assemble     material type:   brick88


**********************************************
comment : put up roof
**********************************************

sub_cover13   assemble     material type:   wood8

sub_cover2    assemble     material type:   wood8

sub_cover1    assemble     material type:   tar_paper2

sub_cover14   assemble     material type:   tar_paper2

cover12       assemble     material type:   shingle12

cover1        assemble     material type:   shingle12
```

```
**********************************************
comment : put up faces for each room
**********************************************

sub_cover11   assemble     material type:  hardboard78

sub_cover6    assemble     material type:  hardboard34

sub_cover4    assemble     material type:  hardboard34

sub_cover5    assemble     material type:  hardboard32

sub_cover3    assemble     material type:  hardboard32

**********************************************
comment : build floor as last step
**********************************************

sub_cover12   assemble     material type:  hard_wood9

**********************************************
comment : put windows in place
**********************************************

window1       complete using pane1          case1

**********************************************
comment : put finish on windows and doors
**********************************************

sill1         finish       paint17         white

door1         finish       paint21         brown
```

```
**********************************************
comment : put on door knobs and hinges
**********************************************

door1       assemble     knob          round32

door1       assemble     hinge         square3in


**********************************************
comment : put final paint on faces
**********************************************

cover10      paint        material type:  paint17

cover3       paint        material type:  paint9

cover5       paint        material type:  paint9

cover2       paint        material type:  paint9

cover4       paint        material type:  paint9

cover11      paint        material type:  paint21
```

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
wood8	$3582	434.194
tar_paper2	$841	6.73333
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

****

* *

Total material cost is $13996

* *

****

Start Raw Materials Report (w/ substitute)

****

* *

sub_cover1: substitute tar_paper1 for tar_paper2

* *

****

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper1	$504	3.36666
wood8	$3582	434.194
tar_paper2	$420	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

****

* *

Total material cost is $14079

* *

****

****

* *

sub_cover1: substitute tar_paper3 for tar_paper2

* *

****

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper3	$370	3.36666
wood8	$3582	434.194
tar_paper2	$420	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

****

* *

Total material cost is $13945

* *

****

****

* *

sub_cover14: substitute tar_paper1 for tar_paper2

* *

****

Raw Materials Report

Item	Cost	Units Required
door1	$16	1
window1	$30	1
concrete1	$1737	347.514
tar_paper2	$420	3.36666
wood8	$3582	434.194
tar_paper1	$504	3.36666
hardboard32	$211	1.54776
hardboard34	$147	1.54722
hardboard78	$200	0.694444
hard_wood9	$900	75
sheath_paper24	$64	0.850277
shingle12	$2020	1616
brick88	$4224	3673.2
paint9	$8	1.0317
paint17	$4	0.551818
paint21	$12	0.923095

****

* *

Total material cost is $14079

* *

****

```
**********************************************************

*                                                        *

 sub_cover14: substitute tar_paper3 for tar_paper2

*                                                        *

**********************************************************

        Raw Materials Report

  Item          Cost       Units Required

door1           $16          1
window1         $30          1
concrete1       $1737        347.514
tar_paper2      $420         3.36666
wood8           $3582        434.194
tar_paper3      $370         3.36666
hardboard32     $211         1.54776
hardboard34     $147         1.54722
hardboard78     $200         0.694444
hard_wood9      $900         75
sheath_paper24  $64          0.850277
shingle12       $2020        1616
brick88         $4224        3673.2
paint9          $8           1.0317
paint17         $4           0.551818
paint21         $12          0.923095

**************************************

*                        *

 Total material cost is $13945

*                        *

**************************************

[ Prolog execution halted ]
```

APPENDIX C - PROTOTYPE PROGRAM LISTINGS

A. PROCESS-ORIENTED PROTOTYPE LISTING

```
code = 4000
project "simulat2"
domains
        file = dat
        l = symbol
        n = integer
        r = real

  include "tdoms.pro"
  include "gdoms.pro"

database
        menuno(row)
        schema(l,row,col,row,col,l)
        schema_object(l,n,n,n,n)
        type(l,l)
        num_props(l,n)
        selected(l)
        design(l,l,l)
        kind_of(l,l)
        opened(l,l)
        saved(l)
        ppdata(l,l,l,l,l,l,l,l,l,l,l,l,l,l,l,l)
        operation(l,l,l,l,l,l,l,l,l,l,n)
        product(l,l,n)
        clock(n)
        ready(n,l,l,n,n,n,n)
        waitng(n,l,l,l,n,n)
        waiting(n,l,l,l,n,n)
        quantity(n)
        working(l,n,n,n,l,n,n)
        least(n,n,l,l,n,n,n)
        dline(n,n)
        resource(l,n,n,n)
```

```
        machine_type(l,l)
        exception(l,l,l)
        pp_except(l,l)
        sched_except(l,l,l)

global Predicates

 determ box(vrow,vcol,vrow,vcol,color,color,fill) - (i,i,i,i,i,i,i) language c
 /* (Row1,Col1,Row2,Col2,LineColor,FillColor,Fill)
 Range for Rows: 0-31999
 Range for Columns: 0-31999
 Fill   = 0  A box will be drawn with color LineColor
          but not filled
        = 1  A box will be drawn with color LineColor
             and filled with color LineColor*/

predicates
        gwrite(row,col,string,color,integer)
        nondeterm repeat
        setEGApalette(integerlist)
        putinlist(integerlist,integer,integer)
        wfs(char)
        wait(n)
        set_pal
        go
        design_phase
        translate1
        process_planning
        translate2
        scheduling
        get_menu(n)
        write_menu(l,color)
        menu(n,l)
        get_mouse_position(n,n)
        action(n,n,n,l)
        highlight(row,color,l)
        color_of(l,color)
        draw_schema
        highlight_type(color,l)
        retract_others
        retract_design
        write_objects(l,l)
```

get_line_no(row)
reset_line_no
create_blanks(n,l)
get_input(l,l)
input_props(l,l)
write_props(l)
change_data(l,l)
input_change(l,l)
write_data(l)
design_data(l,l,l)
retract_assert(l,l,l,l)
load_schema(l)
load_design(l)
save_design(l)
check_quit(l)
check_quit2(l,l)
trans1
pp
produce(l,l,n)
cut(l,n)
cut_top(l,n)
cut_legs(l,n)
brackets(l,n)
brackets_top(l,n)
brackets_legs(l,n)
screw(l,n)
screw_top(l,n)
screw_legs(l,n)
weld(l,n)
weld_top(l,n)
weld_legs(l,n)
assemble(l,n)
assemble_top(l,n)
assemble_legs(l,n)
finish(l,n)
retract_pp
retract_pp_rest
part_one
trans2
check_cut(l,l,l,l,l,l,l,l)
check_screw(l,l,l,l,l,l,l,l)
check_bracket(l,l,l,l,l,l,l,l)

```
        check_weld(l,l,l,l,l,l,l)
        check_assembly(l,l,l,l,l,l,l)
        add_quantities
        add_quant(n,l,l,l,n)
        ret_qty(n)
        print_report
        finished
        still_working
        start
        do_retractc(n)
        available
        can_sched
        do_retractl(n,n,l,l,n,n,n)
        do_retractw(l,n,n,n)
        avail(n,l,l,n,n,n,n)
        avail2(n,l,l,n,n,n,n,n,n)
        get_next
        fig_cost(n,l,l,n,n,n,n)
        part_finished
        do_retractwt(n,l,l)
        retract_duplicates(n,l,l)
        adj_time(l,n,n,n,l,n,n,n)
        check_working(l,n,l)
        retract_sched
        print_working
        check_pp_exceptions(l)
        display_pp_exceptions(l)
        message(l,l,l)
        write_messages
        check_sched_exceptions
        display_sched_exceptions
        remove_windows

include "color.def"
include "cadmouse.pro"

goal
go.
```

```
clauses

color_of(menu,6).
color_of(status,4).
color_of(highlight,15).
color_of(high_sch_text,12).
color_of(schema_text,1).
color_of(schema_box,11).
color_of(schema_conn,6).

go :- repeat,
        remove_windows,
        design_phase,
        remove_windows,
        translate1,
        process_planning,
        translate2,
        scheduling,!.

design_phase :-
        part_one,
        assert(menuno(1)),
        get_menu(1),
        gotowindow(1),
        repeat,
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Select a Command from the Menu Window",4,0),
        get_mouse_position(X,Y),
        action(1,X,Y,C),
        C = "Quit",
        retract_others,
        retract_design,!.
```

```
part_one :-
        text,
        graphics(5,1,0),
        makewindow(1,4,15,"Process Oriented System",0,0,20,54),
        makewindow(2,4,4,"Menu",0,58,11,19),
        makewindow(3,4,13,"Data",13,55,12,25),
        makewindow(4,4,9,"Status",20,0,5,54),
        gotowindow(1),
        set_pal,
        init_mouse,
        show_mouse,
        position_mouse(30,440),!.

translate1 :-
        text,
        graphics(5,1,104),
        makewindow(1,4,15,"Translating Data for Process Planning",0,0,24,80),
        trans1,
        gwrite(1,1,"Hit 'ENTER' to continue",1,0),
        readchar(_),!.

process_planning :-
        text,
        graphics(5,1,104),
        makewindow(1,4,15,"Process Planning",0,0,24,80),
        set_pal,
        consult("pp.dat"),
        gwrite(0,1,"process planning started",7,0),
        ppdata(Name,_,_,_,_,_,_,_,_,_,_,_,_,_),
        check_pp_exceptions(Name),
        pp,
        retract_pp,
        save("process.dat"),
        retract_pp_rest,
        gwrite(10,1,"Hit 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
translate2 :-
        text,
        graphics(5,1,104),
        makewindow(1,4,15,"Translating Data for Scheduling",0,0,24,80),
        trans2,
        gwrite(1,1,"Hit 'ENTER' to continue",1,0),
        readchar(_),!.

scheduling :-
        text,
        graphics(5,1,104),
        makewindow(1,4,15,"Scheduling",0,0,24,80),
        set_pal,
        assert(clock(0)),
        consult("simdata.dta"),
        check_sched_exceptions,
        openwrite(dat,"simtest.doc"),
        repeat,
          writedevice(dat),
          start,
          finished,
        closefile(dat),
        gwrite(16,1,"Scheduling complete - consult simtest.doc for results",1,0),
        gwrite(18,1,"Hit 'ENTER' to continue",4,0),
        readchar(_),
        writedevice(screen),!.

remove_windows :- removewindow,fail,!.
remove_windows :- !.

get_menu(N) :-
        gotowindow(2),
        clearwindow,
        retract(menuno(_)),
        assert(menuno(1)),
        color_of(menu,Color),
        repeat,
          menu(N,X),
          write_menu(X,Color),
          X = "Quit",!.
```

```
write_menu(X,Color) :-
        retract(menuno(R)),
        X1 = X,
        gwrite(R,1,X1,Color,0),
        R1 = R + 1,
        assert(menuno(R1)),!.

get_mouse_position(C,R) :-
        repeat,
        bios(51,reg(3,0,0,0,0,0,0,0),reg(_,Button,Col,Row,_,_,_,_)),
        Button > 0,
        C = (Col / 640) * 80,
        R = (Row / 350) * 24,!.

wfs(C) :- keypressed,readchar(C),!.
wfs(C) :- wait (2000),wfs(C).


wait(0) :- !.
wait(N) :- N1 = N-1, wait(N1).

action(_,X,_,"Continue") :-
        X > 58, X < 77,
        position_mouse(30,440),fail,!.

action(1,X,2,C) :-  X > 58, X < 77,
        color_of(highlight,Color),
        highlight(1,Color,"Load C Schema"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        load_schema(Fname),
        color_of(menu,Mcolor),
        highlight(1,Mcolor,"Load C Schema"),
        C = "Continue",!.
```

```
action(1,X,3,C) :-  X > 58, X < 77,
        opened(design,_),
        color_of(highlight,Color),
        highlight(2,Color,"Load Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"A design data file is already opened",1,0),
        gwrite(2,1,"Press 'ENTER' to continue",4,0),
        readchar(_),
        color_of(menu,Mcolor),
        highlight(2,Mcolor,"Load Design Data"),
        C = "Continue",!.

action(1,X,3,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(2,Color,"Load Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        load_design(Fname),
        color_of(menu,Mcolor),
        highlight(2,Mcolor,"Load Design Data"),
        C = "Continue",!.

action(1,X,4,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(3,Color,"Update Data"),
        get_menu(2),
        repeat,
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Select a Type or 'Quit'",4,0),
        get_mouse_position(X2,Y2),
        action(2,X2,Y2,C2),
        C2 = "Quit",
        get_menu(1),
        C = "Continue",!.
```

```
action(1,X,5,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(4,Color,"Save Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        save_design(Fname),
        color_of(menu,Mcolor),
        highlight(4,Mcolor,"Save Design Data"),
        C = "Continue",!.

action(1,X,7,C) :- X > 58, X < 77,
        check_quit(C),!.

action(2,Y,X,C) :-
        schema_object(Type,Xmin,Ymin,Xmax,Ymax),
        X > Xmin, X < Xmax, Y > Ymin, Y < Ymax,
        assert(selected(Type)),
        position_mouse(30,440),
        color_of(high_sch_text,Color),
        highlight_type(Color,Type),
        repeat,
          get_menu(3),
          gotowindow(4),
          clearwindow,
          gwrite(0,1,"Select a Command from the Menu Window",4,0),
          get_mouse_position(X2,Y2),
          action(3,X2,Y2,C2),
          C2 = "Quit",
        color_of(schema_text,Tcolor),
        highlight_type(Tcolor,Type),
        retract(selected(Type)),
        get_menu(2),
        C = "Continue",!.

action(2,X,5,C) :- X > 58, X < 77,
        C = "Quit",!.
```

```
action(3,X,2,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        gwrite(0,1,"Enter name of object of type",4,0),
        Types = Type,
        gwrite(0,30,Types,1,0),
        str_len(Type,Len),
        Input_pos = 30 + Len,
        gwrite(0,Input_pos," ",9,0),
        readln(Name),
        get_input(Name,Type),
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.

action(3,X,3,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        not(kind_of(_,Type)),
        gwrite(1,1,"No data exists for that type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),
        C = "Continue",!.

action(3,X,3,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        gwrite(0,1,"Enter name of object to change:",4,0),
        gwrite(0,31," ",1,0),
        readln(Name),
        change_data(Name,Type),
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.
```

```
action(3,X,4,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        not(kind_of(_,Type)),
        gwrite(1,1,"No data exists for that type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),
        C = "Continue",!.


action(3,X,4,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        makewindow(3,4,7,"View Data",13,55,12,25),
        repeat,
         kind_of(Object_name,Type),
         gotowindow(3),
         clearwindow,
         reset_line_no,
         gwrite(0,1,"name:",2,0),Objs = Object_name,
         gwrite(0,7,Objs,9,0),
         write_objects(Object_name,Type),
         gotowindow(4),
         clearwindow,
         gwrite(0,1,"Press 'ENTER' to continue or 'q' to quit ",4,0),
         readchar(Quit),
         Quit = 'q',
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.

action(3,X,6,C) :- X > 58, X < 77,
        C = "Quit",!.
```

```
check_quit(C) :-
        saved(_),
        C = "Quit",!.

check_quit(C) :-
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"The design data has not been saved",1,0),
        gwrite(1,1,"Press 's' to save or 'q' to quit ",4,0),
        readln(Quit),
        check_quit2(Quit,C),!.

check_quit2(q,"Quit") :- !.
check_quit2(_,"Continue") :-!.

highlight(Row,Color,Text) :-
        gotowindow(2),Texts = Text,
        gwrite(Row,1,Texts,Color,0),!.

highlight_type(Tcolor,Title) :-
        schema(text,X,Y,_,_,Title),
        str_len(Title,Len),
        Y4 = Y + (8 - (Len / 2)),
        X4 = X,
        gotowindow(1),Titles=Title,
        gwrite(X4,Y4,Titles,Tcolor,0),!.


draw_schema :-
        color_of(schema_text,Tcolor),
        schema(text,X,Y,_,_,Title),
        str_len(Title,Len),
        Y4 = Y + (8 - (Len / 2)),
        X4 = X,
        gotowindow(1),Titles=Title,
        gwrite(X4,Y4,Titles,Tcolor,0),
        X2 = X-1, X3 = X+3, Y2 = Y+2, Y3 = Y+15,
        assert(schema_object(Title,X2,Y2,X3,Y3)),fail,!.
```

```
draw_schema :-
        color_of(schema_box,Bcolor),
        schema(box,X,Y,X1,Y1,_),
        X2 = X, Y2 = Y, X3 = X1, Y3 = Y1,
        box(X2,Y2,X3,Y3,Bcolor,Bcolor,0),fail,!.

draw_schema :-
        color_of(schema_conn,Ccolor),
        schema(conn,X,Y,X1,Y1,_),
        X2 = X, Y2 = Y, X3 = X1, Y3 = Y1,
        line(X2,Y2,X3,Y3,Ccolor),fail,!.

draw_schema.

retract_others :- retract(menuno(_)),fail,!.
retract_others :- retract(schema(_,_,_,_,_,_)),fail,!.
retract_others :- retract(schema_object(_,_,_,_,_)),fail,!.
retract_others :- retract(type(_,_)),fail,!.
retract_others :- retract(num_props(_,_)),fail,!.
retract_others :- retract(selected(_)),fail,!.
retract_others :- retract(opened(_,_)),fail,!.
retract_others :- retract(saved(_)),fail,!.
retract_others :- retract(pp_except(_,_)),fail,!.
retract_others :- !.

retract_design :- retract(design(_,_,_)),fail,!.
retract_design :- retract(kind_of(_,_)),fail,!.
retract_design :- !.

retract_pp :- retract(product(_,_,_)),fail,!.
retract_pp :- retract(exception(_,_,_)),fail,!.
retract_pp :- retract(pp_except(_,_)),fail,!.
retract_pp :- retract(ppdata(_,_,_,_,_,_,_,_,_,_,_,_,_,_)),fail,!.
retract_pp :- !.

retract_pp_rest :- retract(operation(_,_,_,_,_,_,_,_,_)),fail,!.
retract_pp_rest :- !.
```

```
retract_sched :- retract(waiting(_,_,_,_,_,_)),fail,!.
retract_sched :- retract(ready(_,_,_,_,_,_,_)),fail,!.
retract_sched :- retract(dline(_,_)),fail,!.
retract_sched :- retract(machine_type(_,_)),fail,!.
retract_sched :- retract(clock(_)),fail,!.
retract_sched :- retract(resource(_,_,_,_)),fail,!.
retract_sched :- retract(exception(_,_,_)),fail,!.
retract_sched :- retract(sched_except(_,_,_)),fail,!.
retract_sched :- retract(working(_,_,_,_,_,_,_)),fail,!.
retract_sched :- !.

write_objects(Obj,Type) :-
        type(Type,Prop),
        design_data(Obj,Prop,Val),
        get_line_no(N),Props=Prop,
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),
        Write_pos2 = Len + 3,Vals=Val,
        gwrite(N,Write_pos2,Vals,1,0),fail,!.

write_objects(_,_) :- !.

get_input(Name,Type) :-
        kind_of(Name,Type),
        gotowindow(4),
        gwrite(1,1,"An object already exists by that name",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.

get_input(_,_) :-
        retract(saved(_)),fail,!.
```

```
get_input(Name,Type) :-
        assert(kind_of(Name,Type)),
        makewindow(3,4,7,"Add Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        reset_line_no,
        gwrite(0,1,"name:",2,0),
        Objs = Name,
        gwrite(0,7,Objs,9,0),
        write_props(Type),
        reset_line_no,
        input_props(Name,Type),!.

input_props(Name,Type) :-
        type(Type,Prop),Prop  "name",
        Props = Prop,
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter",4,0),
        gwrite(0,7,Props,1,0),
        str_len(Prop,Len),
        Input_pos = Len + 7,
        gwrite(0,Input_pos,": ",4,0),
        readln(Value),
        assert(design(Name,Prop,Value)),
        gotowindow(3),
        Write_pos = Len + 3,
        get_line_no(N),
        Values = Value,
        gwrite(N,Write_pos,Values,1,0),fail,!.

input_props(_,_) :- !.

write_props(Type) :-
        type(Type,Prop),Prop  "name",
        Props = Prop,
        get_line_no(N),
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),fail,!.
```

```
write_props(_) :- !.

change_data(Name,_) :-
        gotowindow(4),
        not(design(Name,_,_)),
        gwrite(1,1,"No object exists by that name",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.

change_data(Name,Type) :-
        gotowindow(4),
        not(kind_of(Name,Type)),
        gwrite(1,1,"Object is the wrong type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.

change_data(_,_) :-
        retract(saved(_)),fail,!.

change_data(Name,Type) :-
        makewindow(3,4,7,"Change Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        reset_line_no,
        gwrite(0,1,"name:",2,0),
        Objs = Name,
        gwrite(0,7,Objs,9,0),
        write_data(Name),
        reset_line_no,
        input_change(Name,Type),!.
```

```
input_change(Name,Type) :-
        type(Type,Prop),Prop  "name",
        Props = Prop,
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter",4,0),
        gwrite(0,7,Props,1,0),
        str_len(Prop,Len),
        Input_pos = Len + 7,
        gwrite(0,Input_pos,": ",4,0),
        readln(Newvalue),
        retract_assert(Name,Prop,Value,Newvalue),
        gotowindow(3),
        Write_pos = Len + 3,
        get_line_no(N),
        str_len(Value,Vlen),
        create_blanks(Vlen,Blank),
        Blanks = Blank,
        gwrite(N,Write_pos,Blanks,1,0),
        Values = Newvalue,
        gwrite(N,Write_pos,Values,1,0),fail,!.

input_change(_,_) :- !.

write_data(Name) :-
        kind_of(Name,Type),
        type(Type,Prop),
        design_data(Name,Prop,Val),Prop  "name",
        Props = Prop,Vals = Val,
        get_line_no(N),
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),
        Write_pos2 = Len + 3,
        gwrite(N,Write_pos2,Vals,1,0),
        fail,!.

write_data(_) :- !.

design_data(Name,Prop,Val) :- design(Name,Prop,Val),!.
```

```
retract_assert(Name,Prop,Value,Newvalue):-
        retract(design(Name,Prop,Value)),
        assert(design(Name,Prop,Newvalue)),!.

load_schema(Fname):-
        opened(schema,Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This schema data has already been loaded",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.


load_schema(Fname):-
        not(existfile(Fname)),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This schema file doesn't exist",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.


load_schema(Fname):-
        consult(Fname),
        assert(opened(schema,Fname)),
        gotowindow(1),
        draw_schema,Fnames = Fname,
        gwrite(0,1,"Schema File:",4,0),
        gwrite(0,14,Fnames,1,0),!.

load_design(Fname):-
        opened(design,Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This design data has already been loaded",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
load_design(Fname) :-
        not(existfile(Fname)),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This data file doesn't exist",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.

load_design(Fname) :-
        existfile(Fname),
        consult(Fname),
        assert(opened(design,Fname)),
        assert(saved(Fname)),
        gotowindow(1),
        gwrite(1,1,"Data File:",4,0),
        Fnames = Fname,
        gwrite(1,14,Fnames,1,0),!.

save_design(Fname) :-
        saved(Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"The design data has already been saved",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.

save_design(Fname) :-
        save("temp.dat"),
        retract_others,
        save(Fname),
        assert(saved(Fname)),
        retract_design,
        consult("temp.dat"),!.

get_line_no(N) :-
        retract(menuno(N)),
        N1 = N + 1,
        assert(menuno(N1)),!.

reset_line_no :-
        retract(menuno(_)),
        assert(menuno(1)),!.
```

```
create_blanks(0,"").
create_blanks(1," ").
create_blanks(N,Blanks) :- N1 = N - 1,
        create_blanks(N1,Nblanks),
        concat(Nblanks," ",Blanks),!.

menu(1,"Load C Schema").
menu(1,"Load Design Data").
menu(1,"Update Data").
menu(1,"Save Design Data").
menu(1,"").
menu(1,"Quit").

menu(2,"").
menu(2,"").
menu(2,"").
menu(2,"Quit").

menu(3,"Add data").
menu(3,"Change data").
menu(3,"View data").
menu(3,"").
menu(3,"Quit").

/* Translator1 Rules follow */

trans1 :- consult("design.dat"),
        kind_of(Name,top),
        assert(product(Name,top,15)),fail,!.
```

```
trans1 :-
        product(Name,_,_),
        design(Name,depth,Dep),
        design(Name,width,Wid),
        design(Name,height,Hgt),
        design(Name,tolerance,Tol),
        design(Name,material,Mat),
        design(Name,finish,Fin),
        kind_of(Name1,connect),
        design(Name1,type,Ctype),
        design(Name1,"x location",X),
        design(Name1,"y location",Y),
        design(Name1,"z location",Z),
        design(Name2,connector,Name1),
        design(Name2,radius,Rad),
        design(Name2,material,Lmat),
        design(Name2,length,Llen),
        assert(ppdata(Name,Dep,Wid,Hgt,Tol,Mat,Fin,Ctype,X,Y,Z,Rad,Lmat,Llen,Name2)),fail,!
.

trans1 :-
        retract_design,
        save("pp.dat"),
        retract_pp,!.


/* Process Planning Rules follow */

pp :- product(Name,Type,Qty),
        produce(Type,Name,Qty),!.

pp :- gwrite(0,1,"PP failed",4,0),!.

produce(top,Name,Qty) :-
        cut(Name,Qty),
        brackets(Name,Qty),
        screw(Name,Qty),
        weld(Name,Qty),
        assemble(Name,Qty),
        finish(Name,Qty),!.
```

```
cut(Name,Qty) :-
        cut_top(Name,Qty),
        cut_legs(Name,Qty),!.

cut(_,_) :- gwrite(1,1,"cut failed",4,0),!.

 cut_top(Name,Qty) :-
        ppdata(Name,Dep,Wid,Hgt,_,Mat,_,_,_,_,_,_,_,_,_),
        assert(operation(Name,tcut,Dep,Wid,Hgt,"0","0","0",Mat,Qty)),!.

 cut_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,_,X,Y,Z,_,Lmat,Llen,Name2),
        assert(operation(Name2,lcut,Llen,"0","0",X,Y,Z,Lmat,Qty)),fail,!.

 cut_legs(_,_) :-!.

brackets(Name,Qty) :-
        brackets_top(Name,Qty),
        brackets_legs(Name,Qty),!.

brackets(_,_) :- gwrite(2,1,"brackets failed",4,0),!.

 brackets_top(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,bracket,_,_,_,_,Lmat,_,_),
        assert(operation(Name,tbracket,"0","0","0","0","0","0",Lmat,Qty)),!.

 brackets_top(_,_) :- !.

 brackets_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,bracket,X,Y,Z,_,Lmat,_,Name2),
        assert(operation(Name2,lbracket,"0","0","0",X,Y,Z,Lmat,Qty)),fail,!.

 brackets_legs(_,_) :-!.

screw(Name,Qty) :-
        screw_top(Name,Qty),
        screw_legs(Name,Qty),!.

screw(_,_) :- gwrite(3,1,"screw failed",4,0),!.
```

```
screw_top(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,screw,_,_,_,_,Lmat,_,_),
        assert(operation(Name,tscrew,"0","0","0","0","0","0",Lmat,Qty)),!.

screw_top(_,_) :- !.

screw_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,screw,X,Y,Z,_,Lmat,_,Name2),
        assert(operation(Name2,lscrew,"0","0","0",X,Y,Z,Lmat,Qty)),fail,!.

screw_legs(_,_) :-!.

weld(Name,Qty) :-
        weld_top(Name,Qty),
        weld_legs(Name,Qty),!.

weld(_,_) :- gwrite(4,1,"weld failed",4,0),!.

weld_top(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,weld,_,_,_,_,Lmat,_,_),
        assert(operation(Name,tweld,"0","0","0","0","0","0",Lmat,Qty)),!.

weld_top(_,_) :- !.

weld_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,weld,X,Y,Z,_,Lmat,_,Name2),
        assert(operation(Name2,lweld,"0","0","0",X,Y,Z,Lmat,Qty)),fail,!.

weld_legs(_,_) :-!.

assemble(Name,Qty) :-
        assemble_top(Name,Qty),
        assemble_legs(Name,Qty),!.

assemble(_,_) :- gwrite(5,1,"assemble failed",4,0),!.

assemble_top(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,bracket,X,Y,Z,_,_,_,_),
        assert(operation(Name,tassemble,"0","0","0",X,Y,Z,bracket,Qty)),fail,!.
```

```
assemble_top(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,screw,X,Y,Z,_,_,_,_),
        assert(operation(Name,tassemble,"0","0","0",X,Y,Z,screw,Qty)),fail,!.

assemble_top(_,_) :- !.

assemble_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,bracket,X,Y,Z,_,_,_,Name2),
        assert(operation(Name2,lassemble,"0","0","0",X,Y,Z,bracket,Qty)),fail,!.

assemble_legs(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,_,screw,X,Y,Z,_,_,_,Name2),
        assert(operation(Name2,lassemble,"0","0","0",X,Y,Z,screw,Qty)),fail,!.

assemble_legs(_,_) :- !.

finish(Name,Qty) :-
        ppdata(Name,_,_,_,_,_,Finish,_,_,_,_,_,_,_,_),
        assert(operation(Name,finish,"0","0","0","0","0","0",Finish,Qty)),!.

finish(_,_) :- gwrite(6,1,"finish failed",4,0),!.


/* Translator2 Rules follow */

trans2 :-
        consult("process.dat"),fail,!.

trans2 :-
        operation(Name,tcut,_,_,X,Y,Z,_,_,_),
        assert(ready(1,Name,sa,0,1,1,1)),
        check_cut(tcut,Name,sa,Loc1,X,Y,Z),
        check_screw(tscrew,Name,Loc1,Loc2,X,Y,Z),
        check_bracket(tbracket,Name,Loc2,Loc3,X,Y,Z),
        check_weld(tweld,Name,Loc3,Loc4,X,Y,Z),
        check_assembly(tassemble,Name,Loc4,_,X,Y,Z),fail,!.
```

```
trans2 :-
        operation(Name,lcut,_,_,X,Y,Z,_,_,_),
        assert(ready(1,Name,sa,0,1,1,1)),
        check_cut(lcut,Name,sa,Loc1,X,Y,Z),
        check_screw(lscrew,Name,Loc1,Loc2,X,Y,Z),
        check_bracket(lbracket,Name,Loc2,Loc3,X,Y,Z),
        check_weld(lweld,Name,Loc3,Loc4,X,Y,Z),
        check_assembly(lassemble,Name,Loc4,_,X,Y,Z),fail,!.

trans2 :-
        add_quantities,
        retract_pp,
        retract_pp_rest,
        save("sched.dat"),!.

check_cut(tcut,Name,Oldl,a,_,_,_) :-
        operation(Name,tcut,_,_,_,_,_,_,_,Qty),
        assert(waitng(1,Name,Oldl,a,1,Qty)),!.
check_cut(tcut,_,Oldl,Oldl,_,_,_) :-!.

check_cut(lcut,Name,Oldl,a,X,Y,Z) :-
        operation(Name,lcut,_,_,X,Y,Z,_,_,Qty),
        assert(waitng(1,Name,Oldl,a,2,Qty)),!.
check_cut(lcut,_,Oldl,Oldl,_,_,_) :- !.

check_screw(tscrew,Name,Oldl,e,_,_,_) :-
        operation(Name,tscrew,_,_,_,_,_,_,_,Qty),
        concat(Oldl,"e",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,e,1,Qty)),!.
check_screw(tscrew,_,Oldl,Oldl,_,_,_) :-!.

check_screw(lscrew,Name,Oldl,e,X,Y,Z) :-
        operation(Name,lscrew,_,_,X,Y,Z,_,_,Qty),
        concat(Oldl,"e",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,e,2,Qty)),!.
check_screw(lscrew,_,Oldl,Oldl,_,_,_) :-!.
```

```
check_bracket(tbracket,Name,Oldl,b,_,_,_) :-
        operation(Name,tbracket,_,_,_,_,_,_,_,Qty),
        concat(Oldl,"b",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,b,1,Qty)),!.
check_bracket(tbracket,_,Oldl,Oldl,_,_,_) :-!.

check_bracket(lbracket,Name,Oldl,b,X,Y,Z) :-
        operation(Name,lbracket,_,_,X,Y,Z,_,_,Qty),
        concat(Oldl,"b",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,b,2,Qty)),!.
check_bracket(lbracket,_,Oldl,Oldl,_,_,_) :-!.

check_weld(tweld,Name,Oldl,c,_,_,_) :-
        operation(Name,tweld,_,_,_,_,_,_,_,Qty),
        concat(Oldl,"c",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,c,1,Qty)),!.
check_weld(tweld,_,Oldl,Oldl,_,_,_) :-!.

check_weld(lweld,Name,Oldl,c,X,Y,Z) :-
        operation(Name,lweld,_,_,X,Y,Z,_,_,Qty),
        concat(Oldl,"c",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,c,2,Qty)),!.
check_weld(lweld,_,Oldl,Oldl,_,_,_) :- !.

check_assembly(tassemble,Name,Oldl,d,_,_,_) :-
        operation(Name,tassemble,_,_,_,_,_,_,_,Qty),
        concat(Oldl,"d",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,d,1,Qty)),!.
check_assembly(tassemble,_,Oldl,Oldl,_,_,_) :-!.

check_assembly(lassemble,Name,Oldl,d,X,Y,Z) :-
        operation(Name,lassemble,_,_,X,Y,Z,_,_,Qty),
        concat(Oldl,"d",Trans),
        assert(waitng(1,Name,Oldl,Trans,0,1)),
        assert(waitng(1,Name,Trans,d,2,Qty)),!.
check_assembly(lassemble,_,Oldl,Oldl,_,_,_) :-!.
```

```
add_quantities :-
        retract(waitng(N,Name,Fr,To,Tool,Qty)),
        assert(quantity(Qty)),
        add_quant(N,Name,Fr,To,Tool),
        retract(quantity(Newq)),
        assert(waiting(N,Name,Fr,To,Tool,Newq)),fail,!.
add_quantities :- !.

  add_quant(N,Name,Fr,To,Tool) :-
        Tool = 0,
        retract(waitng(N,Name,Fr,To,_,_)),
        fail,!.

  add_quant(N,Name,Fr,To,_) :-
        retract(waitng(N,Name,Fr,To,_,Qty)),
        ret_qty(Q),
        Newq = Qty + Q,
        assert(quantity(Newq)),fail,!.

  add_quant(_,_,_,_,_) :- !.

  ret_qty(Q) :- retract(quantity(Q)),!.



/* Scheduling Rules follow */

finished :-
        not(ready(_,_,_,_,_,_)),
        not(waiting(_,_,_,_,_)),
        not(still_working),!.

  still_working :- working(_,_,_,_,_,Time),Time > 0.

start :-
        do_retractc(T),
        T1 = T + 1,
        assert(clock(T1)),
        available,
        print_report,
        part_finished,!.
```

```
  do_retractc(T) :- retract(clock(T)),!.

available :- not(can_sched),!.

available :-
        repeat,
        get_next,
        do_retractl(Cost,P,Name,Mt,Tool,Mn,Time),
        do_retractw(Mt,Mn,0,Cost),
        avail(P,Name,Mt,Tool,Mn,Time,Cost),
        not(can_sched),!.

 can_sched :-
        ready(P,_,c,_,_,_,_),
        not(waiting(P,_,_,c,_,_)),
        working(c,_,_,_,_,_,0),!.
 can_sched :-
        ready(P,_,d,_,_,_,_),
        not(waiting(P,_,_,d,_,_)),
        working(d,_,_,_,_,_,0),!.
 can_sched :-
        ready(_,_,Mt,_,_,_,_),
        Mt <> "c", Mt <> "d",
        working(Mt,_,_,_,_,_,0),!.

 do_retractl(Cost,P,Name,Mt,Tool,Mn,Time) :-
        retract(least(Cost,P,Name,Mt,Tool,Mn,Time)),!.

 do_retractw(_,_,_,9999) :- !.
 do_retractw(Mt,Mn,D,_) :- retract(working(Mt,Mn,_,_,_,_,D)),!.

avail(_,_,_,_,_,_,9999) :- !.
avail(P,Name,Mt,Tool,Mn,Time,_) :-
        avail2(P,Name,Mt,Tool,Mn,Time,Org_quan,Quan,Seq),
        Quan1 = Quan - 1,
        Seq1 = Seq + 1,
        Quan1 > 0,
        assertz(ready(P,Name,Mt,Tool,Org_quan,Quan1,Seq1)),
        retract_duplicates(P,Name,Mt),!.
avail(_,_,_,_,_,_,_) :- !.
```

```
avail2(P,Name,Mt,Tool,Mn,Time,Org_quan,Quan,Seq) :-
        retract(ready(P,Name,Mt,Tool,Org_quan,Quan,Seq)),
        assert(working(Mt,Mn,Tool,P,Name,Seq,Time)),!.

retract_duplicates(P,Name,c) :-
        ready(P,Name2,c,_,_,_,_),
        Name2 <> Name,
        retract(ready(P,Name2,c,_,_,_,_)),fail,!.
retract_duplicates(P,Name,d) :-
        ready(P,Name2,d,_,_,_,_),
        Name2 <> Name,
        retract(ready(P,Name2,d,_,_,_,_)),fail,!.
        retract_duplicates(_,_,_) :- !.

get_next :- assert(least(9999,0,x,x,0,0,0)),
        ready(P,Name,c,Tool,_,_,_),
        not(waiting(P,_,_,c,_,_)),
        fig_cost(P,Name,c,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retractl(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,c,Tool,Mn,Time_req)),
        fail,!.

get_next :-
        ready(P,Name,d,Tool,_,_,_),
        not(waiting(P,_,_,d,_,_)),
        fig_cost(P,Name,d,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retractl(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,d,Tool,Mn,Time_req)),
        fail,!.
```

```
get_next :-
        ready(P,Name,Mt,Tool,_,_,_),
        Mt <> "c", Mt <> "d",
        fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retractl(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,Mt,Tool,Mn,Time_req)),
        fail,!.

get_next :- !.

fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,_,_,0),
        working(Mt,_,_,P,Name,_,Time),Time  0,
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 3,!.

fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,Name,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 2,!.

fig_cost(P,_,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,P,_,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 1,!.

fig_cost(P,_,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,_,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req,!.
```

```
part_finished :- working(Mt,Mn,Tool,P,Name,Seq,Time_left),
        Time_left > 0,
        adj_time(Mt,Mn,Tool,P,Name,Seq,Time_left,New_time),
        New_time = 0,
        not(ready(P,Name,Mt,Tool,_,_,_)),
        check_working(Mt,P,Name),
        do_retractwt(P,Name,Mt),
        fail,!.
part_finished :- !.

  check_working(Mt,P,Name) :-
        working(Mt,_,_,P,Name,_,Time),Time 0,!,fail.
  check_working(_,_,_) :- !.

  do_retractwt(P,_,c) :-
        retract(waiting(P,Name,c,New_mt,New_tool,New_quan)),
        Org_quan = New_quan,
        assert(ready(P,Name,New_mt,New_tool,Org_quan,New_quan,1)),!.

  do_retractwt(P,Name,Mt) :-
        retract(waiting(P,Name,Mt,New_mt,New_tool,New_quan)),
        Org_quan = New_quan,
        assert(ready(P,Name,New_mt,New_tool,Org_quan,New_quan,1)),!.

adj_time(Mt,Mn,Tool,P,Name,Seq,Time_left,New_time) :-
        retract(working(Mt,Mn,Tool,P,Name,Seq,Time_left)),
        New_time = Time_left - 1,
        asserta(working(Mt,Mn,Tool,P,Name,Seq,New_time)),!.

print_report :- nl,clock(Time),
        write("clock period - "),write(Time),nl,
        write("working processes - "),nl,
        not(print_working),!.

print_working :- working(A,B,C,D,N,E,F),F > 0,
        write(D),write(" "),
        write(N),write(" "),
        write(A),write(" "),
        write(C),write(" "),
        write(B),write(" "),
        write(E),write(" "),
        write(F),nl,fail,!.
```

```
/* Process Planning Exception Rules follow */

check_pp_exceptions(_) :-
        consult("ppexcept.dat"),fail,!.

check_pp_exceptions(_) :-
        ppdata(_,_,_,_,Tol,_,_,_,_,_,_,_,_,_,_),
        pp_except(tolerance,Best_tol),
        str_real(Best_tol,Bt),
        str_real(Tol,T),
        T < Bt,
        not(exception(tolerance,Tol,Best_tol)),
        assert(exception(tolerance,Tol,Best_tol)),fail,!.

check_pp_exceptions(_) :-
        ppdata(_,_,_,_,_,_,_,_,_,_,_,Rad,_,_,_),
        pp_except(radius,Bad_rad),
        Rad = Bad_rad,
        not(exception(radius,Rad,Bad_rad)),
        assert(exception(radius,Rad,Bad_rad)),Fail,!.

check_pp_exceptions(Name) :-
        exception(_,_,_),
        display_pp_exceptions(Name),
        retract_pp,
        retract_pp_rest,
        !,fail.

check_pp_exceptions(_) :- !.
```

```
display_pp_exceptions(Name) :-
        makewindow(5,4,15,"Exception Report",3,10,19,60),
        gotowindow(5),
        gwrite(0,25,"Memorandum",1,0),
        gwrite(1,2,"To: Design Department",1,0),
        gwrite(2,2,"From: Process Planning Department",1,0),
        gwrite(3,2,"Subject: Exceptions on design project",1,0),
        Names = Name,
        gwrite(3,40,Names,4,0),
        assert(menuno(5)),
        write_messages,
        retract(menuno(_)),
        gwrite(15,5,"Hit 'ENTER' to continue",4,0),
        readchar(_),!.

write_messages :-
        exception(Type,Val,Std),
        message(Type,Val,Std),
        get_line_no(_),fail,!.
write_messages :- !.

message(tolerance,Tol,Best_tol) :-
        get_line_no(N),
        gwrite(N,2,"The value of the",1,0),
        gwrite(N,19,"tolerance",4,0),
        gwrite(N,29,"for this project is",1,0),
        get_line_no(N1),
        gwrite(N1,2,"too restrictive. A value of ",1,0),
        Bt = Best_tol, T = Tol,
        str_len(Bt,Bt_len),
        gwrite(N1,30,Bt,4,0),
        Pos = 31 + Bt_len,
        gwrite(N1,Pos,"or greater is far less",1,0),
        get_line_no(N2),
        gwrite(N2,2,"costly than the value",1,0),
        gwrite(N2,24,T,4,0),!.
```

```
message(radius,Rad,_) :-
        get_line_no(N),
        gwrite(N,2,"The value of the",1,0),
        gwrite(N,19,"radius",4,0),
        gwrite(N,26,"for this project is",1,0),
        get_line_no(N1),
        gwrite(N1,2,"too expensive. A value of ",1,0),
        gwrite(N1,28,"1.5",4,0),
        gwrite(N1,32,"or",1,0),
        gwrite(N1,35,"1.75",4,0),
        gwrite(N1,40,"is far less",1,0),
        get_line_no(N2),
        gwrite(N2,2,"costly than the value",1,0),
        R = Rad,
        gwrite(N2,24,R,4,0),!.

message(machine,Mt,Msg) :-
        get_line_no(N),
        gwrite(N,2,"The machine you have requested, ",1,0),
        machine_type(Mt,Mname),
        Mts = Mname,
        gwrite(N,34,Mts,4,0),
        get_line_no(N1),
        gwrite(N1,2,"for this project is",1,0),
        get_line_no(N2),
        Msgs = Msg,
        gwrite(N2,2,Msgs,4,0),
        get_line_no(N3),
        gwrite(N3,2,"Please revise your design and resubmit.",1,0),!.


/* Scheduling Exception Rules follow */

check_sched_exceptions :-
        consult("scexcept.dat"),fail,!.

check_sched_exceptions :-
        ready(_,_,Mt,_,_,_,_),
        sched_except(machine,Mt,Msg),
        not(exception(machine,Mt,Msg)),
        assert(exception(machine,Mt,Msg)),fail,!.
```

```
check_sched_exceptions :-
        waiting(_,_,_,Mt,_,_),
        sched_except(machine,Mt,Msg),
        not(exception(machine,Mt,Msg)),
        assert(exception(machine,Mt,Msg)),fail,!.

check_sched_exceptions :-
        exception(_,_,_),
        display_sched_exceptions,
        retract_sched,!,fail.

check_sched_exceptions :- !.

display_sched_exceptions :-
        makewindow(5,4,15,"Exception Report",3,10,19,60),
        gotowindow(5),
        gwrite(0,25,"Memorandum",1,0),
        gwrite(1,2,"To: Design Department",1,0),
        gwrite(2,2,"From: Scheduling Department",1,0),
        gwrite(3,2,"Subject: Exceptions on design project",1,0),
        gwrite(3,40,"top1",4,0),
        assert(menuno(5)),
        write_messages,
        retract(menuno(_)),
        gwrite(15,5,"Hit 'ENTER' to continue",4,0),
        readchar(_),!.

gwrite(R,C,S,Color,0):-
        cursor(R,C),attribute(Color),write(S).
gwrite(_,_,"",_,1):-!.
        gwrite(R,C,S,Color,1):-
        cursor(R,C),attribute(Color),
        frontchar(S,Ch,S1),write(Ch),
        R1=R+1,
        gwrite(R1,C,S1,Color,1).

repeat.
repeat :- repeat.
```

```
setEGApalette(L):-
        X="012345678901234567",
        ptr_dword(X,Segment,Offset),
        putinlist(L,Segment,Offset),
        bios($10,reg($1002,0,0,Offset,0,0,0,Segment),_).

putinlist([],_,_):-!.
putinlist([Byte|T],Segment,Offset):-
        membyte(Segment,Offset,Byte),
        Offset2=Offset+1,
        putinlist(T,Segment,Offset2).
```

B. DATA-ORIENTED PROTOTYPE LISTING

```
code = 3200
project "datadr"
domains
        file = dat
        l = symbol
        n = integer
        r = real

  include "tdoms.pro"
  include "gdoms.pro"

database
        menuno(row)
        schema(l,row,col,row,col,l)
        schema_object(l,n,n,n,n)
        type(l,l)
        num_props(l,n)
        selected(l)
        design(l,l,l)
        kind_of(l,l)
        opened(l,l)
        saved(l)
        product(l,l,n)
        clock(n)
        ready(n,l,l,n,n,n,n)
```

```
        waiting(n,l,l,l,n,n)
        quantity(n)
        working(l,n,n,n,l,n,n)
        least(n,n,l,l,n,n,n)
        dline(n,n)
        resource(l,n,n,n)
        machine_type(l,l)
        machine_used(l,l)
        pp_except(l,l)
        sched_except(l,l,l)
        key(char)
        pcolor(n,n)
        locate(l,n,n)
        trans(l,n,n,n,n,n)

global Predicates

 determ box(vrow,vcol,vrow,vcol,color,color,fill) - (i,i,i,i,i,i,i) language c
 /* (Row1,Col1,Row2,Col2,LineColor,FillColor,Fill)
 Range for Rows: 0-31999
 Range for Columns: 0-31999
 Fill   = 0  A box will be drawn with color LineColor
           but not filled
        = 1  A box will be drawn with color LineColor
           and filled with color LineColor*/

predicates
        gwrite(row,col,string,color,integer)
        nondeterm repeat
        setEGApalette(integerlist)
        putinlist(integerlist,integer,integer)
        wfs(char)
        wfs2
        wait(n)
        set_pal
        go
        design_phase
        scheduling
        get_menu(n)
        write_menu(l,color)
        menu(n,l)
        get_mouse_position(n,n)
```

action(n,n,n,l)
highlight(row,color,l)
color_of(l,color)
draw_schema
highlight_type(color,l)
retract_others
retract_design
write_objects(l,l)
get_line_no(row)
reset_line_no
create_blanks(n,l)
get_input(l,l)
input_props(l,l)
input_props2(l,n,l)
write_props(l)
change_data(l,l)
input_change(l,l)
input_change2(l,n,l)
write_data(l)
design_data(l,l,l)
retract_assert(l,l,l,l)
load_schema(l)
load_design(l)
save_design(l)
check_quit(l)
check_quit2(l,l)
create_pp_data
check_cut(l,l,n,l,l)
check_screw(l,l,n,l,l)
check_bracket(l,l,n,l,l)
check_weld(l,l,n,l,l)
check_assembly(l,l,n,l,l)
retract_pp
retract_pp_rest
part_one
print_report
finished
still_working
start
do_retractc(n)
available
can_sched

```
        do_retractl(n,n,l,l,n,n,n)
        do_retractw(l,n,n,n)
        avail(n,l,l,n,n,n,n)
        avail2(n,l,l,n,n,n,n,n,n)
        get_next
        fig_cost(n,l,l,n,n,n,n)
        part_finished
        do_retractwt(n,l,l)
        retract_duplicates(n,l,l)
        adj_time(l,n,n,n,l,n,n,n)
        check_working(l,n,l)
        retract_sched
        print_working
        message(l,l,l)
        draw_machines
        draw_mach2
        display_ready(n,l,n,n)
        display_working
        display_finished(n,l)
        display_start
        clear_start(n)
        clear_queue(l,n)
        clear_qs(l,n,n)
        clear_mach(l,n)
        draw_queues
        draw_trans(n,n,n,n,n)
        display_trans(l,n,n)
        validate_data(l,l)
        remove_windows

include "color.def"
include "cadmouse.pro"

goal
go.
```

```
clauses

color_of(menu,6).
color_of(status,4).
color_of(highlight,15).
color_of(high_sch_text,12).
color_of(schema_text,1).
color_of(schema_box,11).
color_of(schema_conn,6).

go :- repeat,
        design_phase,
        remove_windows,
        scheduling,!.

design_phase :-
        part_one,
        assert(menuno(1)),
        get_menu(1),
        gotowindow(1),
        repeat,
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Select a Command from the Menu Window",4,0),
        get_mouse_position(X,Y),
        action(1,X,Y,C),
        C = "Quit",
        create_pp_data,
        retract_sched,!.
```

```
part_one :-
        text,
        consult("ppexcept.dat"),
        consult("scexcept.dat"),
        consult("schedinf.dat"),
        graphics(5,1,0),
        set_pal,
        makewindow(1,4,15,"Data Oriented System",0,0,19,54),
        makewindow(2,4,4,"Menu",0,58,11,19),
        makewindow(3,4,13,"Data",13,55,12,25),
        makewindow(4,4,9,"Status",19,0,6,54),
        gotowindow(1),
        init_mouse,
        show_mouse,
        position_mouse(30,440),!.
```

```
scheduling :-
        makewindow(1,104,1,"Shop Floor Simulation",0,0,25,80),
        shiftwindow(1),
        gwrite(2,45,"check-in",12,0),
        gwrite(6,2,"cutting",10,0),
        gwrite(7,2,"machines",10,0),
        gwrite(6,70,"boring",10,0),
        gwrite(7,70,"machines",10,0),
        gwrite(16,2,"welding",10,0),
        gwrite(17,2,"machines",10,0),
        gwrite(16,70,"assembly",10,0),
        gwrite(17,70,"stations",10,0),
        gwrite(22,45,"finished",12,0),
        consult("simdata.dta"),
        consult("schdata.dta"),
        consult("sched.dat"),
        draw_mach2,
        not(draw_queues),
        assert(clock(0)),
        assert(key('s')),
        openwrite(dat,"simtest.doc"),
        repeat,
          writedevice(dat),
          start,
          finished,
        closefile(dat),
        wfs(_),
        writedevice(screen),!.

remove_windows :- removewindow,fail,!.
remove_windows :- !.

get_menu(N) :-
        gotowindow(2),
        clearwindow,
        retract(menuno(_)),
        assert(menuno(1)),
        color_of(menu,Color),
        repeat,
           menu(N,X),
           write_menu(X,Color),
           X = "Quit",!.
```

```
write_menu(X,Color) :- retract(menuno(R)),X1 = X,
        gwrite(R,1,X1,Color,0),
        R1 = R + 1,
        assert(menuno(R1)),!.

get_mouse_position(C,R) :-
        repeat,
        bios(51,reg(3,0,0,0,0,0,0,0),reg(_,Button,Col,Row,_,_,_,_)),
        Button > 0,
        C = (Col / 640) * 80,
        R = (Row / 350) * 24,!.

wfs(C) :- keypressed,readchar(C),!.
wfs(C) :- wait (2000),wfs(C).


wait(0) :- !.
wait(N) :- N1 = N-1, wait(N1).

action(_,X,_,"Continue") :-
        X > 58, X < 77,
        position_mouse(30,440),fail,!.

action(1,X,2,C) :-  X > 58, X < 77,
        color_of(highlight,Color),
        highlight(1,Color,"Load C Schema"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        load_schema(Fname),
        color_of(menu,Mcolor),
        highlight(1,Mcolor,"Load C Schema"),
        C = "Continue",!.
```

```
action(1,X,3,C) :-  X > 58, X < 77,
        opened(design,_),
        color_of(highlight,Color),
        highlight(2,Color,"Load Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"A design data file is already opened",1,0),
        gwrite(2,1,"Press 'ENTER' to continue",4,0),
        readchar(_),
        color_of(menu,Mcolor),
        highlight(2,Mcolor,"Load Design Data"),
        C = "Continue",!.

action(1,X,3,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(2,Color,"Load Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        load_design(Fname),
        color_of(menu,Mcolor),
        highlight(2,Mcolor,"Load Design Data"),
        C = "Continue",!.

action(1,X,4,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(3,Color,"Update Data"),
        get_menu(2),
        repeat,
          gotowindow(4),
          clearwindow,
          gwrite(0,1,"Select a Type or 'Quit'",4,0),
          get_mouse_position(X2,Y2),
          action(2,X2,Y2,C2),
          C2 = "Quit",
        get_menu(1),
        C = "Continue",!.
```

```
action(1,X,5,C) :- X > 58, X < 77,
        color_of(highlight,Color),
        highlight(4,Color,"Save Design Data"),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"Enter the filename: ",4,0),
        readdevice(keyboard),
        readln(Fname),
        save_design(Fname),
        color_of(menu,Mcolor),
        highlight(4,Mcolor,"Save Design Data"),
        C = "Continue",!.

action(1,X,7,C) :- X > 58, X < 77,
        check_quit(C),!.

action(2,Y,X,C) :-
        schema_object(Type,Xmin,Ymin,Xmax,Ymax),
        gotowindow(3),
        clearwindow,
        X > Xmin, X < Xmax, Y > Ymin, Y < Ymax,
        assert(selected(Type)),
        position_mouse(30,440),
        color_of(high_sch_text,Color),
        highlight_type(Color,Type),
        repeat,
          get_menu(3),
          gotowindow(4),
          clearwindow,
          gwrite(0,1,"Select a Command from the Menu Window",4,0),
          get_mouse_position(X2,Y2),
          action(3,X2,Y2,C2),
          C2 = "Quit",
        color_of(schema_text,Tcolor),
        highlight_type(Tcolor,Type),
        retract(selected(Type)),
        get_menu(2),
        C = "Continue",!.

action(2,X,5,C) :- X > 58, X < 77,
        C = "Quit",!.
```

```
action(3,X,2,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        gwrite(0,1,"Enter name of object of type",4,0),
        Types = Type,
        gwrite(0,30,Types,1,0),
        str_len(Type,Len),
        Input_pos = 30 + Len,
        gwrite(0,Input_pos," ",9,0),
        readln(Name),
        get_input(Name,Type),
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.

action(3,X,3,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        not(kind_of(_,Type)),
        gwrite(1,1,"No data exists for that type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),
        C = "Continue",!.

action(3,X,3,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        gwrite(0,1,"Enter name of object to change:",4,0),
        gwrite(0,31," ",1,0),
        readln(Name),
        change_data(Name,Type),
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.
```

```
action(3,X,4,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        not(kind_of(_,Type)),
        gwrite(1,1,"No data exists for that type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),
        C = "Continue",!.


action(3,X,4,C) :- X > 58, X < 77,
        gotowindow(4),
        clearwindow,
        selected(Type),
        makewindow(3,4,7,"View Data",13,55,12,25),
        repeat,
         kind_of(Object_name,Type),
         gotowindow(3),
         clearwindow,
         reset_line_no,
         gwrite(0,1,"name:",2,0),Objs = Object_name,
         gwrite(0,7,Objs,9,0),
         write_objects(Object_name,Type),
         gotowindow(4),
         clearwindow,
         gwrite(0,1,"Press 'ENTER' to continue or 'q' to quit ",4,0),
         readchar(Quit),
         Quit = 'q',
        makewindow(3,4,13,"Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        C = "Continue",!.

action(3,X,6,C) :- X > 58, X < 77,
        C = "Quit",!.

check_quit(C) :-
        saved(_),
        C = "Quit",!.
```

```
check_quit(C) :-
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"The design data has not been saved",1,0),
        gwrite(1,1,"Press 's' to save or 'q' to quit ",4,0),
        readln(Quit),
        check_quit2(Quit,C),!.

check_quit2(q,"Quit") :- !.
check_quit2(_,"Continue") :-!.

highlight(Row,Color,Text) :-
        gotowindow(2),Texts = Text,
        gwrite(Row,1,Texts,Color,0),!.

highlight_type(Tcolor,Title) :-
        schema(text,X,Y,_,_,Title),
        str_len(Title,Len),
        Y4 = Y + (8 - (Len / 2)),
        X4 = X,
        gotowindow(1),Titles=Title,
        gwrite(X4,Y4,Titles,Tcolor,0),!.


draw_schema :-
        color_of(schema_text,Tcolor),
        schema(text,X,Y,_,_,Title),
        str_len(Title,Len),
        Y4 = Y + (8 - (Len / 2)),
        X4 = X,
        gotowindow(1),Titles=Title,
        gwrite(X4,Y4,Titles,Tcolor,0),
        X2 = X-1, X3 = X+3, Y2 = Y+2, Y3 = Y+15,
        assert(schema_object(Title,X2,Y2,X3,Y3)),fail,!.

draw_schema :-
        color_of(schema_box,Bcolor),
        schema(box,X,Y,X1,Y1,_),
        X2 = X, Y2 = Y, X3 = X1, Y3 = Y1,
        box(X2,Y2,X3,Y3,Bcolor,Bcolor,0),fail,!.
```

```
draw_schema :-
        color_of(schema_conn,Ccolor),
        schema(conn,X,Y,X1,Y1,_),
        X2 = X, Y2 = Y, X3 = X1, Y3 = Y1,
        line(X2,Y2,X3,Y3,Ccolor),fail,!.

draw_schema.

retract_others :- retract(menuno(_)),fail,!.
retract_others :- retract(schema(_,_,_,_,_,_)),fail,!.
retract_others :- retract(schema_object(_,_,_,_,_)),fail,!.
retract_others :- retract(type(_,_)),fail,!.
retract_others :- retract(num_props(_,_)),fail,!.
retract_others :- retract(selected(_)),fail,!.
retract_others :- retract(opened(_,_)),fail,!.
retract_others :- retract(saved(_)),fail,!.
retract_others :- retract(sched_except(_,_,_)),fail,!.
retract_others :- retract(machine_used(_,_)),fail,!.
retract_others :- retract(pp_except(_,_)),fail,!.
retract_others :- retract(key(_)),!.
retract_others :- !.

retract_design :- retract(design(_,_,_)),fail,!.
retract_design :- retract(kind_of(_,_)),fail,!.
retract_design :- !.

retract_pp :- retract(product(_,_,_)),fail,!.
retract_pp :- retract(pp_except(_,_)),fail,!.
retract_pp :- !.

retract_pp_rest :- !.

retract_sched :- retract(waiting(_,_,_,_,_,_)),fail,!.
retract_sched :- retract(ready(_,_,_,_,_,_,_)),fail,!.
retract_sched :- retract(dline(_,_)),fail,!.
retract_sched :- retract(machine_type(_,_)),fail,!.
retract_sched :- retract(clock(_)),fail,!.
retract_sched :- retract(resource(_,_,_,_)),fail,!.
retract_sched :- retract(sched_except(_,_,_)),fail,!.
retract_sched :- retract(working(_,_,_,_,_,_,_)),fail,!.
retract_sched :- !.
```

```
write_objects(Obj,Type) :-
        type(Type,Prop),
        design_data(Obj,Prop,Val),
        get_line_no(N),Props=Prop,
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),
        Write_pos2 = Len + 3,Vals=Val,
        gwrite(N,Write_pos2,Vals,1,0),fail,!.

write_objects(_,_) :- !.

get_input(Name,Type) :-
        kind_of(Name,Type),
        gotowindow(4),
        gwrite(1,1,"An object already exists by that name",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.

get_input(_,_) :-
        retract(saved(_)),fail,!.

get_input(Name,Type) :-
        assert(kind_of(Name,Type)),
        makewindow(3,4,7,"Add Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        reset_line_no,
        gwrite(0,1,"name:",2,0),
        Objs = Name,
        gwrite(0,7,Objs,9,0),
        write_props(Type),
        reset_line_no,
        input_props(Name,Type),!.
```

```
input_props(Name,Type) :-
        type(Type,Prop),Prop  "name",
        input_props2(Prop,Len,Value),
        assert(design(Name,Prop,Value)),
        gotowindow(3),
        Write_pos = Len + 3,
        get_line_no(N),
        Values = Value,
        gwrite(N,Write_pos,Values,1,0),fail,!.

input_props(_,_) :- !.

 input_props2(Prop,Len,Value) :-
        Props = Prop,
        repeat,
          gotowindow(4),
          clearwindow,
          gwrite(0,1,"Enter",4,0),
          gwrite(0,7,Props,1,0),
          str_len(Prop,Len),
          Input_pos = Len + 7,
          gwrite(0,Input_pos,": ",4,0),
          readln(Value),
          validate_data(Prop,Value),!.

write_props(Type) :-
        type(Type,Prop),Prop  "name",
        Props = Prop,
        get_line_no(N),
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),fail,!.

write_props(_) :- !.

change_data(Name,_) :-
        gotowindow(4),
        not(design(Name,_,_)),
        gwrite(1,1,"No object exists by that name",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
change_data(Name,Type) :-
        gotowindow(4),
        not(kind_of(Name,Type)),
        gwrite(1,1,"Object is not correct type",1,0),
        gwrite(2,1,"press 'ENTER' to continue",4,0),
        readchar(_),!.

change_data(_,_) :-
        retract(saved(_)),fail,!.

change_data(Name,Type) :-
        makewindow(3,4,7,"Change Data",13,55,12,25),
        gotowindow(3),
        clearwindow,
        reset_line_no,
        gwrite(0,1,"name:",2,0),
        Objs = Name,
        gwrite(0,7,Objs,9,0),
        write_data(Name),
        reset_line_no,
        input_change(Name,Type),!.

input_change(Name,Type) :-
        type(Type,Prop),Prop  "name",
        input_change2(Prop,Len,Newvalue),
        retract_assert(Name,Prop,Value,Newvalue),
        gotowindow(3),
        Write_pos = Len + 3,
        get_line_no(N),
        str_len(Value,Vlen),
        create_blanks(Vlen,Blank),
        Blanks = Biank,
        gwrite(N,Write_pos,Blanks,1,0),
        Values = Newvalue,
        gwrite(N,Write_pos,Values,1,0),fail,!.

input_change(_,_) :- !.
```

```
input_change2(Prop,Len,Newvalue) :-
        Props = Prop,
        repeat,
          gotowindow(4),
          clearwindow,
          gwrite(0,1,"Enter",4,0),
          gwrite(0,7,Props,1,0),
          str_len(Prop,Len),
          Input_pos = Len + 7,
          gwrite(0,Input_pos,": ",4,0),
          readln(Newvalue),
          validate_data(Prop,Newvalue),!.

write_data(Name) :-
        kind_of(Name,Type),
        type(Type,Prop),
        design_data(Name,Prop,Val),Prop "name",
        Props = Prop,Vals = Val,
        get_line_no(N),
        gwrite(N,1,Props,12,0),
        str_len(Prop,Len),
        Write_pos = Len + 1,
        gwrite(N,Write_pos,":",12,0),
        Write_pos2 = Len + 3,
        gwrite(N,Write_pos2,Vals,1,0),
        fail,!.

write_data(_) :- !.

design_data(Name,Prop,Val) :- design(Name,Prop,Val),!.

retract_assert(Name,Prop,Value,Newvalue) :-
        retract(design(Name,Prop,Value)),
        assert(design(Name,Prop,Newvalue)),!.

load_schema(Fname) :-
        opened(schema,Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This schema data has already been loaded",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
load_schema(Fname) :-
        not(existfile(Fname)),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This schema file doesn't exist",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.


load_schema(Fname) :-
        consult(Fname),
        assert(opened(schema,Fname)),
        gotowindow(1),
        draw_schema,Fnames = Fname,
        gwrite(0,1,"Schema File:",4,0),
        gwrite(0,14,Fnames,1,0),!.

load_design(Fname) :-
        opened(design,Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This design data has already been loaded",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.

load_design(Fname) :-
        not(existfile(Fname)),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"This data file doesn't exist",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
load_design(Fname) :-
        existfile(Fname),
        consult(Fname),
        assert(opened(design,Fname)),
        assert(saved(Fname)),
        gotowindow(1),
        gwrite(1,1,"Data File:",4,0),
        Fnames = Fname,
        gwrite(1,14,Fnames,1,0),!.

save_design(Fname) :-
        saved(Fname),
        gotowindow(4),
        clearwindow,
        gwrite(0,1,"The design data has already been saved",1,0),
        gwrite(2,0,"Press 'ENTER' to continue",4,0),
        readchar(_),!.

save_design(Fname) :-
        save("temp.dat"),
        retract_others,
        save(Fname),
        assert(saved(Fname)),
        retract_design,
        consult("temp.dat"),!.

get_line_no(N) :-
        retract(menuno(N)),
        N1 = N + 1,
        assert(menuno(N1)),!.

reset_line_no :-
        retract(menuno(_)),
        assert(menuno(1)),!.

create_blanks(0,"").
create_blanks(1," ").
create_blanks(N,Blanks) :- N1 = N - 1,
        create_blanks(N1,Nblanks),
        concat(Nblanks," ",Blanks),!.
```

```
menu(1,"Load C Schema").
menu(1,"Load Design Data").
menu(1,"Update Data").
menu(1,"Save Design Data").
menu(1,"").
menu(1,"Quit").

menu(2,"").
menu(2,"").
menu(2,"").
menu(2,"Quit").

menu(3,"Add data").
menu(3,"Change data").
menu(3,"View data").
menu(3,"").
menu(3,"Quit").

/* Process Planning Rules follow */

create_pp_data :-
        assert(product(top1,top,15)),fail,!.

create_pp_data :-
        kind_of(Name,top),
        product(Name,_,Qty),
        assert(ready(1,Name,sa,0,1,1,1)),
        check_cut(tcut,Name,Qty,sa,Loc1),
        check_screw(tscrew,Name,Qty,Loc1,Loc2),
        check_bracket(tbracket,Name,Qty,Loc2,Loc3),
        check_weld(tweld,Name,Qty,Loc3,Loc4),
        check_assembly(tassemble,Name,Qty,Loc4,_),fail,!.
```

```
create_pp_data :-
        kind_of(Nme,top),
        product(Nme,_,Qty),
        kind_of(Name,leg),
        assert(ready(1,Name,sa,0,1,1,1)),
        check_cut(lcut,Name,Qty,sa,Loc1),
        check_screw(lscrew,Name,Qty,Loc1,Loc2),
        check_bracket(lbracket,Name,Qty,Loc2,Loc3),
        check_weld(lweld,Name,Qty,Loc3,Loc4),
        check_assembly(lassemble,Name,Qty,Loc4,_),fail,!.

create_pp_data :-
        retract_others,
        retract_design,
        retract_pp,
        retract_pp_rest,
        save("sched.dat"),!.

check_cut(tcut,Name,Qty,Oldl,a) :-
        assert(waiting(1,Name,Oldl,a,1,Qty)),!.
check_cut(tcut,_,_,Oldl,Oldl) :-!.

check_cut(lcut,Name,Qty,Oldl,a) :-
        assert(waiting(1,Name,Oldl,a,2,Qty)),!.
check_cut(lcut,_,_,Oldl,Oldl) :- !.

check_screw(tscrew,Name,Qty,Oldl,e) :-
        design(_,connector,Conn),
        design(Conn,_,screw),
        concat(Oldl,"e",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,e,1,Qty)),!.
check_screw(tscrew,_,_,Oldl,Oldl) :-!.

check_screw(lscrew,Name,Qty,Oldl,e) :-
        design(Name,connector,Conn),
        design(Conn,_,screw),
        concat(Oldl,"e",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,e,2,Qty)),!.
check_screw(lscrew,_,_,Oldl,Oldl) :-!.
```

```
check_bracket(tbracket,Name,Qty,Oldl,b) :-
        design(_,connector,Conn),
        design(Conn,_,bracket),
        concat(Oldl,"b",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,b,1,Qty)),!.
check_bracket(tbracket,_,_,Oldl,Oldl) :-!.

check_bracket(lbracket,Name,Qty,Oldl,b) :-
        design(Name,connector,Conn),
        design(Conn,_,bracket),
        concat(Oldl,"b",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,b,2,Qty)),!.
check_bracket(lbracket,_,_,Oldl,Oldl) :-!.

check_weld(tweld,Name,Qty,Oldl,c) :-
        design(_,connector,Conn),
        design(Conn,_,weld),
        concat(Oldl,"c",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,c,1,Qty)),!.
check_weld(tweld,_,_,Oldl,Oldl) :-!.

check_weld(lweld,Name,Qty,Oldl,c) :-
        design(Name,connector,Conn),
        design(Conn,_,weld),
        concat(Oldl,"c",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,c,2,Qty)),!.
check_weld(lweld,_,_,Oldl,Oldl) :- !.

check_assembly(tassemble,Name,Qty,Oldl,d) :-
        design(_,connector,Conn),
        design(Conn,_,screw),
        concat(Oldl,"d",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,d,1,Qty)),!.
```

```
check_assembly(tassemble,Name,Qty,Oldl,d) :-
        design(_,connector,Conn),
        design(Conn,_,bracket),
        concat(Oldl,"d",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,d,1,Qty)),!.
check_assembly(tassemble,_,_,Oldl,Oldl) :-!.

check_assembly(lassemble,Name,Qty,Oldl,d) :-
        design(Name,connector,Conn),
        design(Conn,_,screw),
        concat(Oldl,"d",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,d,2,Qty)),!.
check_assembly(lassemble,Name,Qty,Oldl,d) :-
        design(Name,connector,Conn),
        design(Conn,_,bracket),
        concat(Oldl,"d",Trans),
        assert(waiting(1,Name,Oldl,Trans,0,1)),
        assert(waiting(1,Name,Trans,d,2,Qty)),!.
check_assembly(lassemble,_,_,Oldl,Oldl) :-!.



/* Scheduling Rules follow */

finished :- not(ready(_,_,_,_,_,_)),
        not(waiting(_,_,_,_,_,_)),
        not(still_working),!.

still_working :- working(_,_,_,_,_,_,Time),Time > 0.
```

```
start :- do_retractc(T),
        T1 = T + 1,
        display_start,
        box(2000,27000,4500,32000,11,11,1),
        writedevice(screen),
        gwrite(2,69,"Time",6,0),
        str_int(Time,T1),
        gwrite(2,74,Time,12,0),
        assert(clock(T1)),
        available,
        print_report,
        writedevice(screen),
        not(display_working),
        wfs2,
        part_finished,!.
 do_retractc(T) :- retract(clock(T)),!.

available :- not(can_sched),!.

available :- repeat,
        get_next,
        do_retractl(Cost,P,Name,Mt,Tool,Mn,Time),
        do_retractw(Mt,Mn,0,Cost),
        avail(P,Name,Mt,Tool,Mn,Time,Cost),
        not(can_sched),!.

 can_sched :- ready(P,_,c,_,_,_,_),
        not(waiting(P,_,_,c,_,_)),
        working(c,_,_,_,_,_,0),!.
 can_sched :- ready(P,_,d,_,_,_,_),
        not(waiting(P,_,_,d,_,_)),
        working(d,_,_,_,_,_,0),!.
 can_sched :- ready(_,_,Mt,_,_,_,_),
        Mt <> "c", Mt <> "d",
        working(Mt,_,_,_,_,_,0),!.

 do_retractl(Cost,P,Name,Mt,Tool,Mn,Time) :-
        retract(least(Cost,P,Name,Mt,Tool,Mn,Time)),!.

 do_retractw(_,_,_,9999) :- !.
 do_retractw(Mt,Mn,D,_) :- retract(working(Mt,Mn,_,_,_,_,D)),!.
```

```
avail(_,_,_,_,_,_,9999) :- !.
avail(P,Name,Mt,Tool,Mn,Time,_) :-
        avail2(P,Name,Mt,Tool,Mn,Time,Org_quan,Quan,Seq),
        Quan1 = Quan - 1,
        Seq1 = Seq + 1,
        display_ready(P,Mt,Org_quan,Seq),
        Quan1 > 0,
        assertz(ready(P,Name,Mt,Tool,Org_quan,Quan1,Seq1)),
        retract_duplicates(P,Name,Mt),!.
avail(_,_,_,_,_,_,_) :- !.

avail2(P,Name,Mt,Tool,Mn,Time,Org_quan,Quan,Seq) :-
        retract(ready(P,Name,Mt,Tool,Org_quan,Quan,Seq)),
        assert(working(Mt,Mn,Tool,P,Name,Seq,Time)),
        clear_qs(Mt,P,Quan),!.

retract_duplicates(P,Name,c) :-
        ready(P,Name2,c,_,_,_,_),
        Name2 <> Name,
        retract(ready(P,Name2,c,_,_,_,_)),fail,!.
retract_duplicates(P,Name,d) :-
        ready(P,Name2,d,_,_,_,_),
        Name2 <> Name,
        retract(ready(P,Name2,d,_,_,_,_)),fail,!.
retract_duplicates(_,_,_) :- !.

get_next :- assert(least(9999,0,x,x,0,0,0)),
        ready(P,Name,c,Tool,_,_,_),
        not(waiting(P,_,_,c,_,_)),
        fig_cost(P,Name,c,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retract1(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,c,Tool,Mn,Time_req)),
        fail,!.
```

```
get_next :-
        ready(P,Name,d,Tool,_,_,_),
        not(waiting(P,_,_,d,_,_)),
        fig_cost(P,Name,d,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retractl(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,d,Tool,Mn,Time_req)),
        fail,!.

get_next :-
        ready(P,Name,Mt,Tool,_,_,_),
        Mt <> "c", Mt <> "d",
        fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req),
        least(X,Y,N,Z,A,B,C),
        Cost < X,
        do_retractl(X,Y,N,Z,A,B,C),
        assert(least(Cost,P,Name,Mt,Tool,Mn,Time_req)),
        fail,!.

get_next :- !.

fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,_,_,0),
        working(Mt,_,_,P,Name,_,Time),Time > 0,
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 3,!.

fig_cost(P,Name,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,Name,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 2,!.

fig_cost(P,_,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,P,_,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req - 1,!.
```

```
fig_cost(P,_,Mt,Tool,Cost,Mn,Time_req) :-
        working(Mt,Mn,_,_,_,_,0),
        dline(P,D),
        resource(Mt,Tool,Mn,Time_req),
        Cost = D * Time_req,!.

part_finished :- working(Mt,Mn,Tool,P,Name,Seq,Time_left),
        Time_left > 0,
        adj_time(Mt,Mn,Tool,P,Name,Seq,Time_left,New_time),
        New_time = 0,
        clear_mach(Mt,Mn),
        not(ready(P,Name,Mt,Tool,_,_,_)),
        check_working(Mt,P,Name),
        clear_qs(Mt,P,1),
        display_finished(P,Mt),
        do_retractwt(P,Name,Mt),
        fail,!.
part_finished :- !.

check_working(Mt,P,Name) :-
        working(Mt,_,_,P,Name,_,Time),Time > 0,!,fail.
check_working(_,_,_) :- !.

do_retractwt(P,_,c) :-
        retract(waiting(P,Name,c,New_mt,New_tool,New_quan)),
        Org_quan = New_quan,
        assert(ready(P,Name,New_mt,New_tool,Org_quan,New_quan,1)),
        display_ready(P,New_mt,Org_quan,1),!.

 do_retractwt(P,Name,Mt) :-
        clear_start(P),
        retract(waiting(P,Name,Mt,New_mt,New_tool,New_quan)),
        Org_quan = New_quan,
        assert(ready(P,Name,New_mt,New_tool,Org_quan,New_quan,1)),
        display_ready(P,New_mt,Org_quan,1),!.

adj_time(Mt,Mn,Tool,P,Name,Seq,Time_left,New_time) :-
        retract(working(Mt,Mn,Tool,P,Name,Seq,Time_left)),
        New_time = Time_left - 1,
        asserta(working(Mt,Mn,Tool,P,Name,Seq,New_time)),!.

draw_mach2 :- not(draw_machines),!.
```

```
draw_machines :- resource(Type,_,Num,_),
        locate(Type,Xpos,Ypos),
        MachY = Ypos + ((Num - 1) * 1500),
        MachX = Xpos,
        ToX = MachX + 1000,
        ToY = MachY + 1000,
        box(MachX,MachY,ToX,ToY,5,5,0),fail,!.

display_ready(P,Mt,Quan,Seq) :-
        locate(Mt,Xpos,Ypos),
        QueueY = Ypos + ((P - 1) * 750), ToY = QueueY + 500,
        QueueX = Xpos - 2000, ToX = QueueX + 1000,
        pcolor(P,Col),Pr= Col,
        box(QueueX,QueueY,ToX,ToY,0,0,1),
        box(QueueX,QueueY,ToX,ToY,Pr,Pr,0),
        NewX1 = QueueX + 4000, NewX2 = ToX + 4000,
        box(NewX1,QueueY,NewX2,Toy,Pr,Pr,0),
        Ratio = (((Seq - 1) * 1000) div Quan) mod 1000,
        Rem = 1000 - Ratio,
        NewX3 = QueueX + Ratio, NewX4 = NewX1 + Rem,
        box(NewX3,QueueY,ToX,ToY,Pr,Pr,1),
        box(NewX4,QueueY,NewX2,ToY,Pr,Pr,1),!.

display_ready(_,_,_,_) :- !.

display_working :- working(Mt,Mn,_,P,_,_,Time),Time > 0,
        display_trans(Mt,P,Time),
        locate(Mt,Xpos,Ypos),
        MachY= Ypos + ((Mn - 1) * 1500) + 250,
        MachX= Xpos + 250,
        ToX = MachX + 500,
        ToY = MachY + 500,
        pcolor(P,Col),Pr= Col,
        box(MachX,MachY,ToX,ToY,1,Pr,1),
        fail,!.

clear_qs(Mt,P,1) :- clear_queue(Mt,P),!.
clear_qs(_,_,_) :- !.
```

```
clear_queue(Type,Num) :-
        display_trans(Type,0,1),
        display_trans(Type,0,2),
        display_trans(Type,0,3),
        locate(Type,Xpos,Ypos),
        MachY = Ypos + ((Num - 1) * 750),
        MachX = Xpos - 2000,
        ToX = MachX + 1000,
        ToY = MachY + 500,
        pcolor(Num,Col),Pr= Col,
        box(MachX,MachY,ToX,ToY,0,0,1),
        box(MachX,MachY,ToX,ToY,Pr,Pr,0),
        NewX1 = MachX + 4000,
        NewX2 = ToX + 4000,
        box(NewX1,MachY,NewX2,ToY,0,0,1),
        box(NewX1,MachY,NewX2,ToY,Pr,Pr,0),!.

clear_queue(_,_) :- !.

clear_mach(Type,Num) :-
        locate(Type,Xpos,Ypos),
        MachY = Ypos + ((Num - 1) * 1500),
        MachX = Xpos,
        ToX = MachX + 1000,
        ToY = MachY + 1000,
        box(MachX,MachY,ToX,ToY,0,0,1),
        box(MachX,MachY,ToX,ToY,5,5,0),!.

clear_mach(_,_) :- !.

draw_queues :- ready(P,_,_,_,_,_,_),
        locate(_,Xpos,Ypos),
        QueueY = Ypos + ((P - 1) * 750), ToY = QueueY + 500,
        QueueX = Xpos - 2000, ToX = QueueX + 1000,
        pcolor(P,Col),Pr= Col,
        box(QueueX,QueueY,ToX,ToY,Pr,Pr,0),
        NewX1 = QueueX + 4000, NewX2 = ToX + 4000,
        box(NewX1,QueueY,NewX2,Toy,Pr,Pr,0),fail,!.
```

```
display_trans(_,_,0) :- !.

display_trans(Type,P,Num) :-
        resource(Type,_,_,Time),N= Time - Num + 1,
        trans(Type,N,FromX,FromY,IncX,IncY),
        draw_trans(FromX,FromY,IncX,IncY,P),!.

display_trans(_,_,_) :- !.

  draw_trans(C,D,E,F,P) :- pcolor(P,Col),Pr= Col,
        C1 = C,D1 = D,E1 = C + 200,F1 = D + 200,
        box(C1,D1,E1,F1,Pr,Pr,1),
        C2 = C1 + E,D2 = D1 + F,E2 = C2 + 200,F2 = D2 + 200,
        box(C2,D2,E2,F2,Pr,Pr,1),
        C3 = C2 + E,D3 = D2 + F,E3 = C3 + 200,F3 = D3 + 200,
        box(C3,D3,E3,F3,Pr,Pr,1),
        C4 = C3 + E,D4 = D3 + F,E4 = C4 + 200,F4 = D4 + 200,
        box(C4,D4,E4,F4,Pr,Pr,1),
        C5 = C4 + E,D5 = D4 + F,E5 = C5 + 200,F5 = D5 + 200,
        box(C5,D5,E5,F5,Pr,Pr,1),!.

display_start :- ready(P,_,sa,_,_,_,_),
        pcolor(P,Col),Pr= Col,
        PosY=13500 + ((P - 1) * 750),ToY= PosY + 500,
        PosX=2500,ToX=PosX + 1000,
        box(PosX,PosY,ToX,ToY,Pr,Pr,1),fail,!.

display_start :- ready(P,_,sb,_,_,_,_),
        pcolor(P,Col),Pr= Col,
        PosY=13500 + ((P - 1) * 750),ToY= PosY + 500,
        PosX=2500,ToX=PosX + 1000,
        box(PosX,PosY,ToX,ToY,Pr,Pr,1),fail,!.

display_start :- ready(P,_,sc,_,_,_,_),
        pcolor(P,Col),Pr= Col,
        PosY=13500 + ((P - 1) * 750),ToY= PosY + 500,
        PosX=2500,ToX=PosX + 1000,
        box(PosX,PosY,ToX,ToY,Pr,Pr,1),fail,!.
```

```
display_start :- ready(P,_,sd,_,_,_,_),
        pcolor(P,Col),Pr= Col,
        PosY=13500 + ((P - 1) * 750),ToY= PosY + 500,
        PosX=2500,ToX=PosX + 1000,
        box(PosX,PosY,ToX,ToY,Pr,Pr,1),fail,!.

display_start :- !.

clear_start(P) :-
        PosY=13500 + ((P - 1) * 750) - 1,ToY= PosY + 502,
        PosX=2500,ToX=PosX + 1002,
        box(PosX,PosY,ToX,ToY,0,0,1),!.

display_finished(P,d) :-
        pcolor(P,Col),Pr= Col,
        PosY=13500 + ((P - 1) * 750),ToY= PosY + 500,
        PosX=28500,ToX=PosX + 1000,
        box(PosX,PosY,ToX,ToY,Pr,Pr,1),!.

display_finished(_,_) :- !.

wfs2 :- not(keypressed),key('g'),!.
wfs2 :- keypressed,readchar(C),
        key(X),retract(key(X)),assert(key(C)),!.
wfs2 :- wait (2000),wfs2.

print_report :-
        writedevice(dat),
        nl,clock(Time),
        write("clock period - "),write(Time),nl,
        write("working processes - "),nl,
        not(print_working),
        writedevice(screen),!.

print_working :- working(A,B,C,D,N,E,F),F > 0,
        write(D),write(" "),
        write(N),write(" "),
        write(A),write(" "),
        write(C),write(" "),
        write(B),write(" "),
        write(E),write(" "),
        write(F),nl,fail,!.
```

```
/* Exception Rules follow */

validate_data(tolerance,Val) :-
        pp_except(tolerance,Best_tol),
        str_real(Best_tol,Bt),
        str_real(Val,V),
        V < Bt,
        message(tolerance,Val,Best_tol),!,fail.

validate_data(radius,Rad) :-
        pp_except(radius,Bad_rad),
        str_real(Rad,R),
        str_real(Bad_rad,B),
        R = B,
        message(radius,Rad,Bad_rad),!,fail.

validate_data(type,Type) :-
        machine_used(Type,Mach),
        sched_except(machine,Mach,Msg),
        message(machine,Mach,Msg),!,fail.

validate_data(_,_) :- !.

message(tolerance,Tol,Best_tol) :-
        gotowindow(4),
        clearwindow,
        gwrite(0,2,"The value of the",1,0),
        gwrite(0,19,"tolerance",4,0),
        gwrite(0,29,"for this project is too",1,0),
        gwrite(1,2,"restrictive. A value of ",1,0),
        Bt = Best_tol, T = Tol,
        str_len(Bt,Bt_len),
        gwrite(1,26,Bt,4,0),
        Pos = 27 + Bt_len,
        gwrite(1,Pos,"or greater is far",1,0),
        gwrite(2,2,"less costly than the value",1,0),
        gwrite(2,29,T,4,0),
        gwrite(3,2,"Press 'ENTER' to continue",4,0),
        readchar(_),!.
```

```
message(radius,Rad,_) :-
        gotowindow(4),
        clearwindow,
        gwrite(0,2,"The value of the",1,0),
        gwrite(0,19,"radius",4,0),
        gwrite(0,26,"for this project is",1,0),
        gwrite(1,2,"too expensive. A value of ",1,0),
        gwrite(1,28,"1.5",4,0),
        gwrite(1,32,"or",1,0),
        gwrite(1,35,"1.75",4,0),
        gwrite(1,40,"is far less",1,0),
        gwrite(2,2,"costly than the value",1,0),
        R = Rad,
        gwrite(2,24,R,4,0),
        gwrite(3,2,"Press 'ENTER' to continue",4,0),
        readchar(_),!.

message(machine,_,Msg) :-
        gotowindow(4),
        clearwindow,
        gwrite(0,2,"The machine required for this operation is ",1,0),
        Msgs = Msg,
        gwrite(1,2,Msgs,4,0),
        gwrite(2,2,"Please revise your design accordingly.",1,0),
        gwrite(3,2,"Press 'ENTER' to continue",4,0),
        readchar(_),!.


gwrite(R,C,S,Color,0):-
        cursor(R,C),attribute(Color),write(S).
gwrite(_,_,"",_,1):-!.
        gwrite(R,C,S,Color,1):-
        cursor(R,C),attribute(Color),
        frontchar(S,Ch,S1),write(Ch),
        R1=R+1,
        gwrite(R1,C,S1,Color,1).

repeat.
repeat :- repeat.
```

```
setEGApalette(L):-
        X="0123456789012345678",
        ptr_dword(X,Segment,Offset),
        putinlist(L,Segment,Offset),
        bios($10,reg($1002,0,0,Offset,0,0,0,Segment),_).

putinlist([],_,_):-!.
putinlist([Byte|T],Segment,Offset):-
        membyte(Segment,Offset,Byte),
        Offset2=Offset+1,
        putinlist(T,Segment,Offset2).
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor C. Thomas Wu, Code 52Wq Department of Computer Science Naval Postgraduate School Monterey, California 93943	5
4.	Professor David L. Smith, Code 69Sm Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943	1
5.	Professor Robert B. McGhee, Code 52Mz Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
6.	Professor David K. Hsiao, Code 52Hq Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
7.	Professor John R. Ward, Code 62Wa Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1

		No. Copies
8.	Professor Bruno O. Shubert, Code 55Sy Department of Operations Research Naval Postgraduate School Monterey, California 93943	1
9.	Professor Mandula Waldron Department of Engineering Graphics Ohio State University 2070 Neil Avenue Columbus, Ohio 43210	1
10.	Lt. Relle Lyman Naval Sea Systems Command Sea 90G Washington, D.C. 20362	1
11.	Major Dana E. Madison Academy of Health Sciences, U.S. Army ATTN: Health Care Administration Division Fort Sam Houston, Texas 78234-6100	3